OPTIMASI PENJADWALAN MATA PELAJARAN PADA KURIKULUM 2013 DENGAN MENGGUNAKAN HIBRIDISASI ALGORITME GENETIKA DAN *SIMULATED ANNEALING* (STUDI KASUS: SMA NEGERI 6 SURABAYA)

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Priscillia Vinda Gunawan
NIM: 145150201111097

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

OPTIMASI PENJADWALAN MATA PELAJARAN PADA KURIKULUM 2013 DENGAN MENGGUNAKAN HIBRIDISASI ALGORITME GENETIKA DAN *SIMULATED ANNEALING* (STUDI KASUS: SMA NEGERI 6 SURABAYA)

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Priscillia Vinda Gunawan
NIM: 145150201111097

Skripsi ini telah diuji dan dinyatakan lulus pada
29 Juni 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Imam Cholissodin, S.Si, M.Kom.	Bayu Rahayudi , S.T, M.T.
NIK: 201201 850719 1 001	NIP: 19740712 200604 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 23 Mei 2018

Priscillia Vinda Gunawan

NIM: 145150201111097

KATA PENGANTAR

Segala puja dan puji serta syukur Alhamdulillah Penulis panjatkan kepada Allah SWT yang telah melimpahkan rahmat, karunia, serta hidayah-Nya sehingga penulis dapat menyelesaikan skripsi dengan judul: "**Optimasi Penjadwalan Mata Pelajaran Kurikulum 2013 Dengan Menggunakan Hibridisasi Algoritme Genetika dan *Simulated Annealing* (Studi Kasus: SMA Negeri 6 Surabaya)".**

Skripsi ini merupakan salah satu syarat kelulusan yang harus ditempuh di Fakultas Ilmu Komputer, Program Studi Informatika Universitas Brawijaya Malang. Dan tak lupa pula, penulis juga mengucapkan terimakasih yang sebesar-besarnya kepada pihak-pihak yang telah memberikan bantuan selama pengerjaan skripsi ini dari awal hingga terselesaikannya laporan skripsi ini, diantaranya:

1. Wayan Firdaus Mahmudy, S.Si., M.T., Ph.D., Ir. Heru Nurwasito, M.Kom., Suprapto, S.T, M.T., Edy Santoso, S.Si, M.Kom. selaku Dekan, Wakil Dekan I, Wakil Dekan II, Wakil Dekan III Fakultas Ilmu Komputer Universitas Brawijaya Malang.
2. Imam Cholissodin, S.Si, M.Kom., selaku dosen pembimbing 1 dan Bayu Rahayudi , S.T, M.T., selaku dosen pembimbing 2 yang telah memberikan bimbingan, saran, serta arahan selama penyusunan skripsi ini.
3. Seluruh dosen Fakultas Ilmu Komputer yang telah mendidik dan memberikan ilmu serta wawasannya selama menempuh pendidikan dan menyelesaikan skripsi ini.
4. Seluruh civitas akademika Fakultas Ilmu Komputer Universitas Brawijaya yang telah banyak memberikan bantuan serta dukungan kepada penulis selama menempuh pendidikan dan menyelesaikan skripsi ini.
5. Gunawan dan Rustianah selaku kedua orang tua penulis yang telah mendukung penulis dari awal pendidikan hingga menyelesaikan skripsi ini baik secara moril dan materiil, serta mendukung, memberi semangat melalui setiap doa dan kasih sayangnya yang tulus tiada henti.
6. Reggy Linda Gunawan dan Surya Setiawan Putra selaku kedua adik penulis yang telah memberikan semangat serta dukungannya kepada penulis dalam menyelesaikan skripsi ini.
7. Richa Amalia Permatasari dan Wanda Athira Luqyana selaku teman, sahabat, dan saudara bagi penulis selama masa perkuliahan yang telah meluangkan waktu, tenaga, pikiran, serta kasih sayang yang tulus untuk bergaul dengan penulis, mengajarkan banyak hal yang belum penulis ketahui, dan selalu ada di saat suka maupun duka.
8. Romario Siregar selaku partner penulis selama perkuliahan yang telah memberikan semangat dan dukungannya dalam menyelesaikan skripsi ini.
9. Muhammad Mishbahul Munir dan Bahruddin El Hayat selaku teman asisten praktikum dan teman seperjuangan skripsi yang telah lulus, yang

telah banyak membantu, memberikan motivasi, dan meluangkan waktu dalam pengerjaan skripsi ini.

10. Eni, Dinda, Puspita, Reyna, Heidi, Rara, Eva selaku teman-teman Bendungan Bening 49 yang telah memberi dukungan dan menemani hari-hari penulis selama di perantauan Kota Malang.
11. Teman-teman BEMTIIK Bersatu III dan teman-teman satu kepanitiaan dengan penulis, terimakasih atas pengalaman organisasi dan kepanitiaan selama ini.
12. Seluruh teman-teman Informatika UB angkatan 2014 serta semua pihak yang tidak dapat penulis sebutkan satu persatu yang telah membantu dan mendukung penulis selama pendidikan sampai terselesaikannya skripsi ini baik secara langsung maupun tidak langsung.

Penulis menyadari bahwasannya skripsi ini masih mempunyai kekurangan. Oleh karena itu, segala bentuk kritik dan saran yang membangun sangat penulis harapkan. Dan besar harapan penulis skripsi ini dapat bermanfaat bagi semua pihak yang membaca, berkepentingan dan khususnya bagi penulis sendiri.

Malang, 23 Mei 2018

Penulis
priscilliavinda14@gmail.com

ABSTRAK

Priscillia Vinda Gunawan, Optimasi Penjadwalan Mata Pelajaran pada Kurikulum 2013 dengan menggunakan Hibridisasi Algoritme Genetika dan *Simulated Annealing* (Studi Kasus: SMA Negeri 6 Surabaya).

Pembimbing: Imam Cholissodin, S.Si, M.Kom. dan Bayu Rahayudi , S.T, M.T.

Penjadwalan merupakan salah satu masalah komputasi yang tidak mudah diselesaikan. Dalam menyelesaikannya harus disiapkan secara sistematis, dengan memaksimalkan sumber daya dan waktu yang ada secara efektif dan efisien. Masalah penjadwalan dapat terjadi dalam berbagai bidang, tidak terkecuali bidang pendidikan. SMA Negeri 6 Surabaya merupakan salah satu sekolah menengah atas di Surabaya yang memiliki permasalahan dalam menemukan slot waktu yang tepat dengan jumlah guru yang terbatas serta terdapat beberapa *constraint* yang harus dipenuhi dalam penjadwalan mata pelajaran. Salah satu metode yang dapat digunakan dalam penjadwalan mata pelajaran adalah dengan menggunakan hibridisasi algoritme genetika dan *simulated annealing* (GA-SA) karena GA memiliki kelemahan konvergensi dini dan kemungkinan terjebak dalam *local* optimum, maka diberikan SA sebagai solusi untuk menutupi kelemahan GA dan mampu bertahan pada *local* optimum. Proses hibridisasi algoritme ini dilakukan dengan langkah pertama pada GA menggunakan representasi kromosom bilangan integer, *one-cut point crossover*, *reciprocal exchange mutation*, dan *elitism selection*. Pada langkah yang kedua, dilakukan proses *simulated annealing* dengan menggunakan *neighborhood move*. Hasil yang diberikan adalah penjadwalan mata pelajaran dengan memenuhi *constraint* yang ada. Berdasarkan penelitian yang dilakukan, didapatkan parameter-parameter optimal yaitu jumlah generasi 270, jumlah populasi 90, kombinasi nilai *cr* dan *mr* adalah 0.3 dan 0.7 dengan rata-rata nilai *fitness* sebesar 0,999000.

Kata kunci: *optimasi, penjadwalan, mata pelajaran, hibridisasi, algoritme genetika, simulated annealing*

ABSTRACT

Priscillia Vinda Gunawan, Optimization of Course Scheduling in Curriculum 2013 by using Genetic Algorithm and Simulated Annealing Hybridization (Case Study: SMA Negeri 6 Surabaya).

Advisor: Imam Cholissodin, S.Si, M.Kom. and Bayu Rahayudi , S.T, M.T.

Scheduling is one of the computational problems that is not easily solved. For solving it must be prepared systematically, by maximizing resources and time available effectively and efficiently. Scheduling problems can occur in various fields, including education. SMA Negeri 6 Surabaya is one of the high schools in Surabaya that has problems finding the right time slot with a limited number of teachers and there are some constraints that must be met in the scheduling of subjects. One method that can be used in the scheduling of subjects is to use genetic algorithm hybridization and simulated annealing (GA-SA) because GA has the weakness of early convergence and possibly stuck in local optimum, then SA is given as a solution to cover the weakness of GA and able to survive on a local optimum. The algorithm hybridization process is done with the first step in GA using chromosome representation of integer numbers, one-cut point crossover, reciprocal exchange mutation, and elitism selection. In the second step, a simulated annealing process is done using neighborhood move. The results given are scheduling the subjects by meeting the existing constraints. Based on the research, the optimum parameters are the number of generation 270, the population 90, the combination of cr and mr are 0.3 and 0.7 with the average fitness value of 0.999000.

Keywords: *optimization, scheduling, subjects, hybridization, genetic algorithms, simulated annealing*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR PERSAMAAN

DAFTAR KODE PROGRAM

BAB 1 PENDAHULUAN

1.1 Latar belakang

Salah satu masalah komputasi yang sulit untuk diselesaikan adalah penjadwalan. Penjadwalan ini memiliki tujuan yang penting untuk menemukan slot waktu yang tepat untuk beberapa tugas ataupun pekerjaan dengan sumber daya manusia yang terbatas. Masalah penjadwalan seperti ini banyak terjadi di berbagai bidang, tidak terkecuali penjadwalan dalam bidang pendidikan. Permasalahan yang terjadi pada bidang pendidikan ini terkait dengan sulitnya dalam membangun sebuah jadwal yang tepat sesuai dengan preferensi administrasi, guru, serta siswa (Benli & Botsali, 2004). Penjadwalan tersebut dapat dikatakan optimal apabila dapat memenuhi batasan-batasan tertentu. Aturannya terdapat dua buah batasan, yaitu batasan yang harus dipenuhi atau biasa disebut dengan *hard constraint* dan batasan yang tidak harus dipenuhi atau biasa disebut dengan *soft constraint*. Apabila *hard constraint* tidak terpenuhi maka solusi tersebut tidak dapat dikatakan optimal, namun jika terdapat pelanggaran *soft constraint* masih dapat ditekankan untuk mendapatkan penjadwalan yang optimal (Sari, et al., 2015).

SMA Negeri 6 Surabaya merupakan salah satu sekolah menengah atas di Surabaya yang memiliki permasalahan mengenai penjadwalan mata pelajaran, yang mana SMA Negeri 6 Surabaya memiliki 24 kelas, 3 ruang lab, 2 jurusan, dan 19 mata pelajaran. SMA Negeri 6 memiliki total waktu 45 jam pelajaran dalam satu minggu dengan total guru sejumlah 44 guru. Akan tetapi hanya 41 guru saja yang masuk dalam penjadwalan mata pelajaran. Dari pemaparan tersebut, proses penyusunan jadwal secara manual dinilai kurang efektif karena membutuhkan waktu yang lama, seperti halnya SMA Negeri 6 Surabaya yang membutuhkan waktu hingga 1 minggu setiap tahunnya dalam menyelesaikan penyusunan jadwal mata pelajaran dan penyusunan jadwal tersebut dapat berubah pada tiap semester karena adanya pergantian formasi guru. Disamping jumlah kelas, jumlah jam, jumlah guru, dan jumlah mata pelajaran yang menjadi faktor penyusunan jadwal mata pelajaran terdapat faktor lain yang membuat penyusunan jadwal menjadi kompleks. Antara lain waktu pengerjaan yang tidak konsisten, terdapat guru yang mengajar dengan 2 mata pelajaran yang berbeda, dan guru dengan jabatan tertentu memiliki porsi mengajar yang berbeda.

Oleh sebab itu diperlukan penyusunan jadwal mata pelajaran yang sesuai. Misalnya mata pelajaran pendidikan jasmani dan kesehatan (penjaskes) harus dilakukan pada pagi hari atau di jam pertama pada saat pembelajaran berlangsung. Sebelumnya dalam proses pembuatan mata pelajaran dilakukan dengan menggunakan *tools* Microsoft Excel, yang telah dirancang untuk mendeteksi jadwal yang bentrok dan juga kebutuhan porsi mengajar setiap guru. Kebutuhan porsi mengajar tersebut dimasukkan dan dibuat secara manual oleh bagian waka kurikulum, Ety Yuniandari, S.Pd. dengan dibantu satu staf waka kurikulum, Dra. Ika Mustikawati, M.pd. Penyusunan mata pelajaran di SMA

Negeri 6 Surabaya tidak dapat menggunakan *timetable* (tabel waktu) karena ada beberapa kondisi yang kurang sesuai dengan kondisi di SMA Negeri 6 Surabaya. Oleh sebab itu perlu adanya sebuah sistem penjadwalan mata pelajaran di SMA Negeri 6 Surabaya.

Salah satu cabang dalam ilmu *Evolution Algorithms* (EA) adalah algoritme genetika. Algoritme genetika ini dapat digunakan dalam mengatasi permasalahan optimasi. Karena algoritme genetika merupakan metode untuk menyelesaikan permasalahan optimasi dengan model matematika yang kompleks (Mahmudy, 2013). Namun algoritme genetika masih memiliki beberapa keterbatasan, yang artinya hasil yang dihasilkan kurang optimum apabila diterapkan pada permasalahan yang luas (Elhaddad & Sallabi, 2010). Sehingga diperlukan kombinasi dengan metode lain atau biasa disebut dengan istilah hibridisasi, yang mana hibridisasi ini diperlukan untuk menghindari konversi dini dan memperkuat kemampuan algoritme genetika.

Salah satu metode hibridisasi yang digunakan adalah *simulated annealing* (SA). Penerapan SA di dalam algoritme genetika menggunakan metode solusi yaitu *neighborhood move* untuk menemukan solusi baru di setiap iterasi SA yang dimodifikasi. *Simulated annealing* mampu mendapatkan hasil mendekati optimal pada masalah yang kompleks, karena ruang pencarian yang luas dan mampu diterapkan pada masalah kombinatorial (Lin, et al., 2011). Proses hibridisasi ini dapat membantu menyeimbangkan kemampuan algoritme genetika dalam mengeksplorasi pencarian dan mengeksploitasi daerah *local* dalam pencarian solusi yang lebih baik (Mahmudy, et al., 2013). Seperti pada penelitian sebelumnya yang dilakukan oleh Avicena (2016) dalam optimasi penjadwalan pengawas ujian semester menggunakan hibridisasi algoritme genetika dan *simulated annealing* untuk studi kasus Fakultas Ilmu Komputer Universitas Brawijaya yang mendapatkan hasil mendekati optimal.

Berdasarkan penjelasan permasalahan di SMA Negeri 6 Surabaya, maka kombinasi antara algoritme genetika dan *simulated annealing* dapat digunakan sebagai solusi dalam topik skripsi yang diangkat oleh penulis dengan judul "Optimasi Penjadwalan Mata Pelajaran Pada Kurikulum 2013 dengan Menggunakan Hibridisasi Algoritme Genetika dan *Simulated Annealing* (Studi Kasus: SMA Negeri 6 Surabaya)". Pada penelitian diharapkan dapat menyelesaikan permasalahan penjadwalan mata pelajaran di SMA Negeri 6 Surabaya. Sehingga mencapai tujuan untuk mendapatkan solusi optimal dalam meningkatkan efektifitas dan efisiensi dalam penjadwalan mata pelajaran.

1.2 Rumusan masalah

Berdasarkan latar belakang masalah yang telah diuraikan di atas, maka dapat dirumuskan permasalahannya sebagai berikut:

1. Bagaimana mengimplementasikan hibridisasi algoritme genetika dan *simulated annealing* untuk mengoptimasi penjadwalan mata pelajaran di SMA Negeri 6 Surabaya?

2. Berapa tingkat kualitas solusi yang dihasilkan dari hibridisasi algoritme genetika dan *simulated annealing* untuk studi kasus penjadwalan mata pelajaran di SMA Negeri 6 Surabaya?

1.3 Tujuan

Tujuan dari peneltian ini adalah sebagai berikut:

1. Menerapkan hibridisasi algoritme genetika dan *simulated annealing* dalam mencari solusi optimal untuk studi kasus SMA Negeri 6 Surabaya.
2. Mengukur seberapa tinggi tingkat kualitas solusi yang dihasilkan dari hibridisasi algoritme genetika dan *simulated annealing* untuk studi kasus di SMA Negeri 6 Surabaya.

1.4 Manfaat

Manfaat yang dapat diperoleh dari penelitian ini adalah sebagai berikut:

1. Mendapatkan solusi optimal dalam penjadwalan mata pelajaran di SMA Negeri 6 Surabaya
2. Membuat jadwal mata pelajaran di SMA Negeri 6 Surabaya menjadi lebih efektif dan efisien.

1.5 Batasan masalah

Agar penelitian ini fokus pada permasalahan yang telah dirumuskan, maka diperlukan batasan masalah dalam penelitian ini:

1. Objek data yang digunakan hanya didapat dari SMA Negeri 6 Surabaya.
2. Parameter-parameter yang terlibat dalam penjadwalan adalah jumlah jam pelajaran di setiap mata pelajaran pada setiap kelas, jumlah jam pelajaran pada setiap guru, serta kode unik masing-masing guru dalam setiap mata pelajaran.
3. Parameter-parameter hibridisasi algoritme genetika dan *simulated annealing* yang digunakan adalah *popSize* (ukuran populasi), jumlah generasi maksimal, *crossover rate*, *mutation rate*, suhu awal (*Tc*), suhu akhir (*Ta*), konstanta penurunan suhu (*β*), serta metode modifikasi solusi pada SA menggunakan *neighborhood move*.

1.6 Sistematika pembahasan

Sistematika penulisan pada penelitian ini terdapat tujuh bab sebagai berikut:

BAB 1 PENDAHULUAN

Berisi tentang latar belakang dilaksanakannya penelitian, identifikasi masalah yang akan diselesaikan, rumusan masalah yang akan dibahas, tujuan dan manfaat yang ingin dicapai, batasan masalah yang ditentukan, dan sistematika pembahasan dari Optimasi Penjadwalan Mata Pelajaran

Pada Kurikulum 2013 dengan Menggunakan Hibridisasi Algoritme Genetika dan *Simulated annealing* (Studi Kasus: SMA Negeri 6 Surabaya).

BAB 2 LANDASAN KEPUSTAKAAN

Berisi referensi dan dasar teori dilakukannya penelitian yang berhubungan dengan optimasi, penjadwalan mata pelajaran, dan metode hibridisasi. Serta algoritme genetika, dan *simulated annealing* yang dapat digunakan untuk mendukung Optimasi Penjadwalan Mata Pelajaran Pada Kurikulum 2013 dengan Menggunakan Hibridisasi Algoritme Genetika dan *Simulated annealing* (Studi Kasus: SMA Negeri 6 Surabaya).

BAB 3 METODOLOGI

Berisi metode yang digunakan dalam penelitian, seperti studi pustaka, analisis kebutuhan, *survey*/wawancara, pengumpulan data, pengolahan data, perancangan sistem, implementasi, pengujian, analisis, dan kesimpulan dalam Optimasi Penjadwalan Mata Pelajaran Pada Kurikulum 2013 dengan Menggunakan Hibridisasi Algoritme Genetika dan *Simulated annealing* (Studi Kasus: SMA Negeri 6 Surabaya).

BAB 4 PERANCANGAN

Bab ini membahas tentang perancangan antarmuka, perancangan pengujian sistem, dan evaluasi dari Optimasi Penjadwalan Mata Pelajaran Pada Kurikulum 2013 dengan Menggunakan Hibridisasi Algoritme Genetika dan *Simulated annealing* (Studi Kasus: SMA Negeri 6 Surabaya).

BAB 5 IMPLEMENTASI

Berisi implementasi yang telah dilakukan yaitu spesifikasi sistem, implementasi algoritme, dan implementasi *interface* dari Optimasi Penjadwalan Mata Pelajaran Pada Kurikulum 2013 dengan Menggunakan Hibridisasi Algoritme Genetika dan *Simulated annealing* (Studi Kasus: SMA Negeri 6 Surabaya).

BAB 6 PENGUJIAN DAN ANALISIS

Berisi hasil pengujian dan analisis dari sistem. Bab ini menjelaskan proses pengujian dan analisis yang telah dilakukan dalam Optimasi Penjadwalan Mata Pelajaran Pada Kurikulum 2013 dengan Menggunakan Hibridisasi Algoritme Genetika dan *Simulated annealing* (Studi Kasus: SMA Negeri 6 Surabaya).

BAB 7 PENUTUP

Berisi kesimpulan dan saran dari penelitian. Saran dapat berisi kritik dan saran untuk pengembangan selanjutnya dan kesimpulan didapat dari serangkaian penelitian yang telah dilakukan dalam Optimasi Penjadwalan Mata Pelajaran Pada Kurikulum 2013 dengan Menggunakan Hibridisasi Algoritme Genetika dan *Simulated annealing* (Studi Kasus: SMA Negeri 6 Surabaya).

BAB 2 LANDASAN KEPUSTAKAAN

Landasan kepustakaan berisi tentang kajian pustaka dan teori yang mendukug penelitian. Pada kajian pustaka membahas tentang penelitian-penelitian yang sudah dilakukan sebelumnya sebagai acuan dalam pelaksanaan penelitian berikutnya. Untuk dasar teori pada penelitian ini difokuskan penjadwalan mata pelajaran.

2.1 Kajian Pustaka

Kajian pustaka berikut ini berisi tentang penelitian-penelitian sebelumnya yang terkait dengan topik yang diangkat oleh penulis. Penelitian-penelitian tersebut dapat dilihat pada Tabel 2.1.

Tabel 2.1 Tinjauan Pustaka

No	Pustaka	Objek	Metode	Hasil
1	(Efendi, et al., 2017)	Jadwal Mata Pelajaran	Algoritme Genetika	Nilai *fitness* yang dihasilkan mendekati optimal (bernilai 1), yaitu 0.8451.
2	(Sofianti, 2004)	*Multipurpose Batch Chemical Plant*	Algoritme Genetika – *Simulated Annealing*	GA-SA dapat diterapkan dalam penjadwalan *multipurpose batch chemical* secara leluasa dan hasilnya lebih baik dibanding GA/SA saja.
3	(Avicena, 2016)	Jadwal Pengawas Ujian Semester	Hibridisasi GA-SA	Teknik hibridisasi GA-SA mampu menyelesaikan permasalahan. Menghasilkan nilai *fitness* sebesar 0.941.
4	(Iman, et al., 2018)	Komposisi Makanan Untuk Ibu Hamil	*Hybrid* Algoritme Genetika dan *Simulated Annealing*	Sistem rekomendasi dengan *hybrid* Algoritme Genetika dan *Simulated Annealing* dapat

				memenuhi batas toleransi sebesar ±10%.
5	(Al-Milli, 2011)	*University Course Timetabling*	*Hybrid Genetic Algorithms With Simulating Annealing*	Menghasilkan jadwal yang layak dan berkualitas baik. Serta memberikan hasil yang konsisten.
6	(Norozi, et al., 2012)	*Multi-objective Sequencing Problem in High-product Mix Shop-floor*	*A hybrid* GA-SA	GA-SA terbukti Mampu mencapai solusi yang lebih baik di sebagian besar masalah.

Beberapa hasil penelitian yang digunakan sebagai kajian pustaka, antara lain penelitian yang dilakukan oleh Efendi (2017) dalam mengoptimasi penjadwalan mata pelajaran menggunakan algoritme genetika pada studi kasus SMK Negeri 2 Kediri menghasilkan nilai fitness sebesar 0.8451 dengan jumlah populasi terbaik sebanyak 90, nilai kombinasi *cr* dan *mr* sebesar 0.5 dan 0.5, serta jumlah generasi sebanyak 40000. Kromosom yang dibangkitkan berasal dari data penugasan guru yang direpresentasikan dengan bilangan *biner*. Setiap bilangan biner berisi kode penugasan yang *random*. Metode reproduksi yang digunakan adalah *one-cut point crossover* dan *reciprocal exchange mutation.* Serta proses seleksi nilai *fitness* menggunakan *elitism selection*.

Penelitian yang dilakukan oleh Sofianti (2004) dalam penjadwalan *multipurpose batch chemical plant* menggunakan algoritme genetika-*simulated annealing* mendapatkan hasil mendekati optimal, yaitu *makespan* terkecil dengan waktu komputasi yang lebih cepat dengan cara perhitungan yang lebih mudah. *Simulated annealing* digunakan untuk mengendalikan penjadwalan pada saat penurunan suhu agar dapat bertahan dalam kondisi *local* optimum. Salah satu jenis mutasi yang digunakan adalah probabilitas adaptif yang dikendalikan oleh suhu, artinya tingkat mutasi diatur untuk pencarian *neighbourhood* yang baik. Penjadwalan dapat dilakukan secara leluasa dengan mengggunakan GA-SA karena perfomansi perhitungan tidak terganggu karena dapat disesuaikan dengan fungsinya.

Penelitian yang dilakukan oleh Avicena (2016) mengenai penjadwalan pengawas ujian semester dengan menggunakan hibridisasi algoritme genetika dan *simulated annealing* pada studi kasus Fakultas Ilmu Komputer Universitas Brawijaya menggunakan hibridisasi GA-SA dengan metode *one-cut point* pada proses *crossover*, menggunakan *random mutation* pada proses mutasi, *elitism selection* pada proses seleksi, dan metode modifikasi solusi pada SA menggunakan *neighborhood move*. Data yang digunakan adalah data UAS ganjil

tahun 2015/2016 sejumlah 561 ujian dalam kurun waktu 10 hari, parameter *popSize* yang bernilai 95, jumlah generasi sebesar 1000, nilai *cr* sebesar 0.5, nilai *mr* sebesar 0.5, *β* bernilai 0.9, *Tc* bernilai 1000, dan *Ta* bernilai 10 mampu memiliki nilai *fitness* sebesar 0.941 dengan waktu komputasi selama 43 menit.

Penelitian yang dilakukan oleh Iman (2018) dalam mengoptimasi komposisi makanan untuk ibu hamil menggunakan *hybrid* algoritme genetika dan *simulated annealing* menghasilan sistem rekomendasi makanan yang dapat memenuhi batas toleransi sebesar ± 10% yang telah ditetapkan oleh ahli gizi dengan parameter terbaik yang digunakan adalah jumlah populasi sebesar 2900, jumlah generasi sebesar 220, nilai *cr* sebesar 0.4, nilai *mr* sebesar 0.6, temperatur awal (*T0*) sebesar 1, dan nilai alpha sebesar 0.7. Penyelesaian masalah ini menggunakan metode *one-cut point crossover*, *reciprocal exchange mutation*, dan *elitism selection*.

Penelitian yang dilakukan oleh Al-Milli (2011) mengangkat permasalahan mengenai penjadwalan kursus di universitas, yang mana permasalahan tersebut merupakan permasalahan *non polynomial-hard*, artinya permasalahan sulit diselesaikan dengan mode kovensional dan waktu komputasi yang lama. Permasalahan pada penjadwalan memberikan sejumlah *N* mata kuliah yang dijadwalkan dalam 5 hari dan 9 petemuan disetiap mata kuliah dengan *T timeslot* sejumlah 45, *R* ruangan, *M* mahasiswa, dan kebutuhan fasilitas. Mekanisme seleksi dilakukan dengan memilih nilai *fitness* tertinggi, proses *crossover* menggunakan *single point*, mekanisme mutasi dilakukan dengan menukar satu atau lebih gen, dan menggunakan nilai *threshold* sebagai nilai *random* dalam proses *simulated annealing*. Meskipun *output* hanya menghasilkan dua hasil terbaik, akan tetapi *hybrid* GA-SA ini mampu memberikan *output* yang layak dan konsisten di semua batasan masalah.

Selanjutnya penelitian yang dilakukan oleh Norozi (2012) mengenai masalah *Multi-Objective Sequencing* dalam *High-Product Mix Shop-Floor* menghasilkan solusi yang lebih baik pada sebagian masalah, akan tetapi akan lebih baik lagi jika diterapkan prosedur yang lebih sistematis dan jumlah parameter *input* yang lebih banyak. Penelitian ini menggunakan jumlah populasi yang besar agar menghasilkan keberagaram individu dan mencegah terjadinya konvergensi dini. Proses yang dilakukan pada penelitian ini antara lain, inisialisasi populasi, *tournament selection*, *partially mapped crossover*, *swap mutation*, *neighborhood search structure* (NSS), penurunan suhu pada *simulated annealing*, dan yang terakhir *elitism* untuk menemukan solusi terbaik dari proses *hybrid*.

2.2 Penjadwalan

Penjadwalan merupakan kegiatan yang mengalokasikan sejumlah kegiatan, dapat berupa pertemuan, sekolah, kuliah, ujian, dan sebagainya ke dalam sebuah variabel waktu dengan kebutuhan tempat yang terbatas serta sedapat mungkin memenuhi *constraint* yang ada. Seiring dengan berkembangnya kebutuhan manusia, muncul sebuah permasalahan dalam penjadwalan tersebut.

Permasalahan dapat terjadi di berbagai bidang, seperti pendidikan, sosial, industri, maupun di bidang lainnya. Penjadwalan memiliki tujuan untuk menemukan slot waktu yang sesuai dengan terbatasnya sumber daya. Dalam permasalahan ini, penjadwalan dapat memiliki tujuan yang berbeda-beda tergantung dengan batasan dari permasalahan yang diangkat (Benli & Botsali, 2004).

2.2.1 Penjadwalan Mata Pelajaran

Penjadwalan yang akan dibahas adalah penjadwalan mengenai mata pelajaran pada kurikulum tahun 2013. Kurikulum 2013 merupakan kurikulum yang dapat mengintegrasikan kemampuan, tema, konsep, dan topik dalam satu disiplin ilmu atau beberapa disiplin ilmu pada seluruh peserta didik (Poerwati & Amri, 2013). Kurikulum 2013 dikembangkan dari Kurikulum 2006 (KTSP) yang dilandasi dengan pemikiran abad ke-21, perkembangan pengetahuan yang modern, knowledge-based society, dan kompetensi masa depan (Gultom, 2013).

Penjadwalan mata pelajaran ini merupakan salah satu jenis *timetable*. Permasalahan *timetable* digolongkan sebagai *NP-Hard Problem (non-deterministic polynomial time)*, artinya waktu yang digunakan untuk seluruh kombinasi solusi merupakan waktu yang polinom di mana meningkatnya waktu berbanding lurus dengan banyaknya variabel yang digunakan. Sehingga permasalahan penjadwalan ini akan membutuhkan waktu yang lama jika digunakan algoritme kovensional (Darmawan & Hasibuan, 2014).

2.3 Algoritme Genetika

Algoritme Genetika (GA) merupakan salah satu cabang dari *Evolutionary Algorithm* (EA) yang pertama kali dicetuskan oleh John Holland pada tahun 1975. Algoritme genetika banyak digunakan dalam *artificial intelligence* untuk teknik optimasi. Algoritme ini bersifat iteratif, probabilistik, dan dapat menyesuaikan diri dalam pencarian untuk optimasi (Wen & Wen-jun, 2006). Dasar dari algoritme genetika ini adalah untuk menerapkan teori evolusi genetika yang dicetuskan oleh Darwin dalam bentuk komputasi. Setiap individu dapat mengalami perubahan genetik (mutasi) untuk menyesuaikan diri dengan lingkungannya dalam bertahan hidup (Gendreau & Potvin, 2010).

Di dalam algoritme genetika terdapat sekumpulan individu yang berkumpul menjadi sebuah populasi. Setiap individu dari populasi tersebut dapat didefinisikan sebagai solusi dari permasalahan. Penggunaan algoritme genetika dalam menyelesaikan permasalahan, harus merancang sebuah kode yang sesuai. Dalam mengimplementasikan algoritme genetika digunakan beberapa operator, misalnya dalam proses reproduksi melibatkan operator *crossover* (pindah silang) dan *mutation* (mutasi), dalam proses evolusi melibatkan proses *selection* (seleksi). Fungsi *fitness* pada setiap individu digunakan untuk mengetahui seberapa bagus kualitas solusi yang dihasilkan dari individu tersebut (Sivanandam & Deepa, 2008).

2.3.1 Istilah Dasar dalam Algoritme Genetika

Terdapat beberapa definisi penting dalam Algoritme Genetika yang perlu diperhatikan, yaitu:

1. Gen (*Genotype*) merupakan variabel dasar pembentuk kromosom. Berisi sebuah informasi yang dapat dinyatakan dalam bentuk biner, float, integer, karakter, atau kombinatorial.
2. *Allele* merupakan nilai dari gen.
3. Kromosom atau individu merupakan gabungan dari beberapa gen yan membentuk suatu nilai tertentu. Nilai tersebut merupakan nilai dari solusi permasalahan yang diangkat.
4. Populasi merupakan sekumpulan individu yang akan diproses bersama dalam satu siklus proses evolusi.
5. Generasi menyatakan satu siklus proses evolusi atau satu iterasi di dalam algoritme genetika.
6. Iterasi adalah bilangan yang menunjukkan tingkat generasi dan perulangan.
7. Nilai *fitness* menyatakan seberapa baik nilai dari suatu individu atau solusi yang dihasilkan, dan nilai inilah yang dijadikan acuan untuk mencari nilai optimal dari suatu permasalahan.

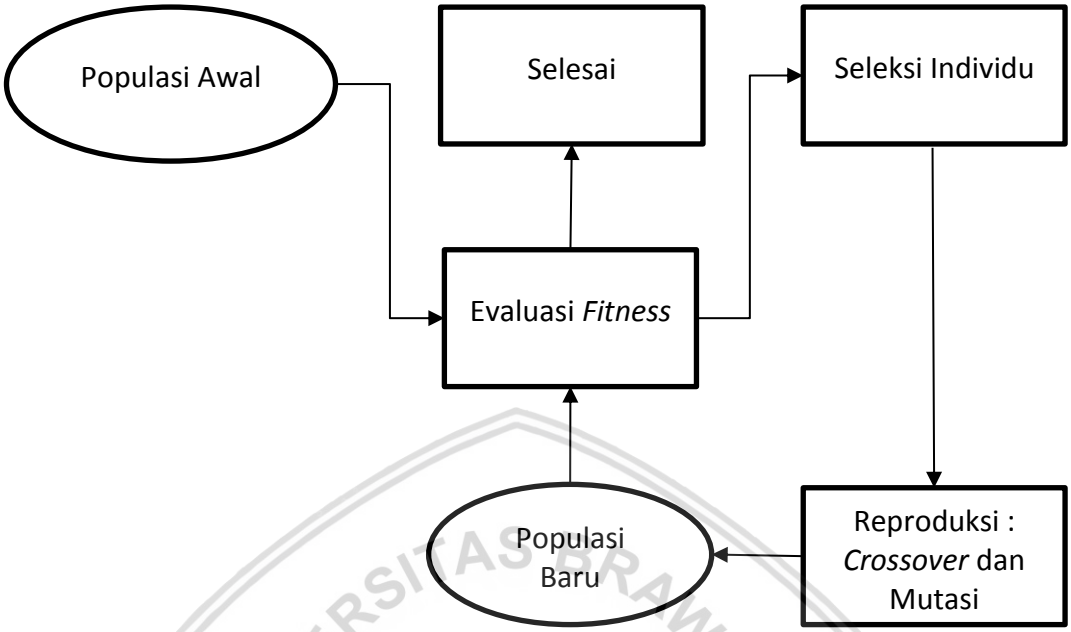
2.3.2 Siklus Algoritme Genetika

Perlu adanya suatu pengkodean solusi *(encoding)* dalam menyelesaikan suatu permasalahan dengan menggunakan algoritme genetika menjadi sebuah kromosom. Kromosom terdiri dari sejumlah gen yang merepresentasikan sejumlah variabel keputusan yang akan membentuk sebuah solusi. Dan *encoding* tersebut menentukan tingkat kelayakan sebuah solusi (Mahmudy, 2013).

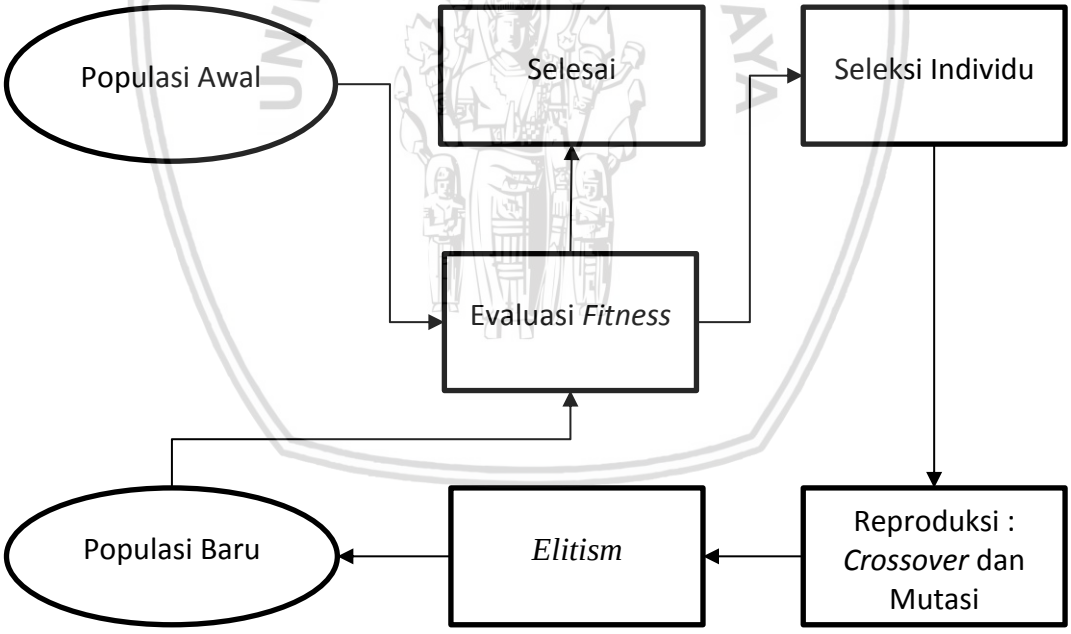
Pada Gambar 2.1 merupakan Siklus Algoritme genetika pertama kali dikenalkan oleh David Goldberg. Kemudian siklus ini diperbaiki oleh beberapa ilmuwan yang mengembangkan algoritme genetika, yaitu Zbigniew Michalewicz dengan menambahkan operator *elitism* dan membalik proses seleksi setelah proses reproduksi.

Pada Gambar 2.2 pencarian individu secara bersamaan pada algoritme genetika disebut dengan populasi, dimasa disetiap individu terdapat kromosom yang merepresentasikan sebuah solusi. Kromosom terdiri dari sejumlah gen yang mewakili variabel tertentu untuk membentuk sebuah solusi, yang mana nilai dari gen tersebut disebut dengan *allele*. Individu populasi awal dibangun secara *random* lalu dievaluasi untuk menemukan nilai *fitness* disetiap individu. Reproduksi yang dilakukan oleh individu induk (*parent*) akan menghasilkan inividu baru yang disebut anak (*child*). Terdapat dua operator reproduksi dalam algoritme genetika, yaitu *crossover* yang menggabungkan sifat dua buah individu yang menghasilkan *child* dan mutasi yang memodifikasi dari sebuah individu.

Generasi populasi dilakukan terus menerus sampai solusi yang di dapat telah konvergen (Mawaddah & Mahmudy, 2006).



Gambar 2.1 Siklus Algoritme Genetika oleh David Goldberg



Gambar 2.2 Siklus Algoritme Genetika yang diperbarui oleh Michalewicz

2.3.3 Komponen dalam Algoritme Genetika

2.3.3.1 Inisialisasi Awal

Hal pertama yang harus dilakukan terlebih dahulu adalah mendefinisikan nilai ukuran populasi atau biasa disebut *popSize*. Kemudian membangkitkan populasi awal atau menginisialisasikannya. *PopSize* merupakan jumlah individu (kromosom) yang terdapat dalam populasi (Mahmudy, 2013).

Dalam menentukan hal ini terdapat beberapa macam pendekatan, tetapi hal yang paling mendasar adalah pertimbangan antara efisiensi, efektifitas, serta banyaknya jumlah populasi awal. Karena dengan banyaknya jumlah populasi tersebut akan berpengaruh pada banyaknya individu yang akan di proses sehingga membutuhkan waktu komputasi yang lama. Akan tetapi apabila jumlah populasi sedikit akan mengakibatkan konvergensi dini meskipun waktu komputasi cukup singkat. Dengan jumlah populasi awal yang optimal akan menghasilkan efisiensi dan efektifitas yang baik (Glover & Kochenberger, 2003)

2.3.3.2 Reproduksi

Proses reproduksi menghasilkan *child* dari dua *parent* yang disilangkan. Pada proses ini dapat menggunakan dua operator genetika, yaitu *crossover* dan *mutation*. Untuk menghasilkan *child* maka perlu menentukan *crossover rate* (*cr*), yaitu rasio *child* yang dihasilkan pada proses *crossover* berdasarkan ukuran populasi. Selain itu juga perlu menentukan *mutation rate* (*mr*), yaitu rasio *child* yang dihasilkan pada proses *mutation* berdasarkan ukuran populasi (Mahmudy, 2013).

***Crossover* (Kombinasi Silang)**

Merupakan proses mengambil solusi dari dua buah *parent* yang menghasilkan *child* (Sivanandam & Deepa, 2008). Berikut contoh metode *crossover*:

1. *One-cut-point crossover*

Dalam tabel 2.2 dilakukan pemilihan satu titik potong secara *random* serta mengganti bagian kanan setiap *parent* untuk mendapatkan *child* (Mahmudy, 2013).

Tabel 2.2 *One-cut-point crossover*

Titik potong (t1)	2					
Parent Pertama	1	0	1	1	0	1
Parent Kedua	1	1	0	0	1	1
Child	1	0	0	0	1	1

2. *Two-cut-point crossover*

Dalam tabel 2.3 dilakukan pemilihan dua titik potong secara *random*. Gen pertama sampai dengan titik potong pertama pada *parent* pertama dimasukkan ke dalam *child*, kemudian gen dari titik potong pertama sampai dengan titik potong kedua dari *parent* kedua dimasukkan lagi ke dalam *child* (lanjutan dari gen sebelumnya), dan lanjutan gen yang terakhir dari *child* diambil dari gen dari titik potong kedua sampai gen terakhir dari *parent* pertama.

Tabel 2.3 *Two-cut-point crossover*

Titik potong 1 (t1)	2					
Titik potong 2 (t2)	4					
Parent Pertama	1	0	1	1	0	1
Parent Kedua	1	1	0	0	1	1
Child	1	0	0	0	0	1

***Mutation* (Mutasi)**

Merupakan proses mengubah sebagian nilai gen (*allele*) dari suatu individu. Berikut beberapa contoh operasi mutasi (Adamanti, 2002):

1. Mutasi Pengkodean Biner

Pada tabel 2.4 pengkodean biner *allele* hanya bernilai 0 atau 1, *allele* gen yang dipilih *random* akan di *inverse* nilainya.

Tabel 2.4 Mutasi Pengkodean Biner

Gen terpilih	4					
Kromosom *parent*	1	1	0	1	1	0
Kromosom *child*	1	1	0	0	1	0

2. Mutasi Pengkodean Permutasi

Pada tabel 2.5 pengkodean dilakukan dengan cara menukarkan *allele* pada dua posisi gen yang berbeda.

Tabel 2.5 Pengkodean Permutasi

Gen terpilih	2,5					
Kromosom *parent*	1	2	3	4	5	6
Kromosom *child*	1	5	3	4	2	6

Untuk representasi permutasi, sudah ada beberapa operator mutasi yang diciptakan seperti:

a. Inversion Mutation

Memilih secara *random* dua titik/posisi dalam kromosom, lalu menginversikan substring di antara kedua titik/posisi tersebut.

b. Insertion Mutation

Memilih secara *random* satu buah gen, lalu dimasukkan ke dalam kromosom juga secara *random*.

c. *Displacement Mutation*

Memilih secara *random* sebagian/sekelompok gen, lalu dimasukkan ke dalam kromosom juga secara *random*.

d. *Reciprocal Exchange Mutation* (REM)

Memilih secara *random* dua titik/posisi dalam kromosom, lalu menukar kedua titik/posisi tersebut.

e. *Tree Encoding*

Memilih node yang akan diubah.

f. *Value Encoding* (Mutasi Pengkodean Nilai)

Pada tabel 2.6 dilakukan dengan memilih salah satu gen kemudian menambah atau mengurangi nilainya dengan angka *random* yang kecil. Untuk mendapatkan nilai pengganti digunakan rumus *x'=x+r(max-min)*, yang mana *x'* = nilai pengganti, *x* = *allele* terpilih, *r* = nilai *random* antara -1 sampai dengan 1, *max* = nilai batas terbesar, dan *min* = nilai batas terkecil.

Tabel 2.6 Pengkodean Nilai

Gen terpilih	4					
Angka *random*	-0,56					
Kromosom *parent*	2,21	3,14	1,29	9,76	1,51	9,55
Kromosom *child*	2,21	3,14	1,29	9,20	1,51	9,55

2.3.3.3 Evaluasi

Pada proses ini dilakukan perhitungan nilai *fitness* pada tiap kromosom. Apabila nilai *fitness* yang dihasilkan besar, maka dapat menggunakan solusi dari kromosom dengan nilai *fitness* terbesar tersebut (Mahmudy, 2013). Untuk menghitung nilai *fitness* pada tiap individu dapat menggunakan Persamaan (2.1) di bawah ini:

$$fitness = \frac{1}{(1+f(x))} \quad (2.1)$$

Keterangan :

f(x) = Fungsi objektif (total nilai pelanggaran) dari individu x

x = Individu yang dihitung nilai *fitness*-nya

Nilai *f(x)* yang didapat merupakan total nilai pelanggaran pada setiap indvidu. Total nilai tersebut berasal dari keseluruhan jumlah pelanggaran setiap *constraint* dikali dengan bobot pelanggarannya. Dapat dilihat pada Persamaan (2.2) di bawah ini:

$$f(x) = \sum_{k=1}^{n}(JmlPenalti_k * BobotPenalti_k) \quad (2.2)$$

Keterangan:

$JmlPenalti_k$ = Jumlah pelanggaran setiap *constraint k*

$BobotPenalti_k$ = Bobot pelanggaran setiap *constraint k*

k = Indeks *contraint*

n = Banyaknya *constraint*

2.3.3.4 Seleksi

Proses ini dilakukan untuk menentukan individu yang dipertahankan hidupnya untuk generasi selanjutnya dari himpunan populasi dan *child*. Apabila nilai *fitness* pada kromosomnya tinggi, maka peluangnya semakin besar (Mahmudy, 2013).

```
procedure AlgoritmeGenetika
begin
t=0
inisialisasi P(t)
while (bukan kondisi berhenti) do
      reproduksi C(t) dari P(t)
      evaluasi P(t) dan C(t)
      seleksi P(t+1) dari P(t) dan C(t)
      t=t+1
end while
end
```

Gambar 2.3 *Pseudocode* Algoritme Genetika
Sumber: (Gen & Cheng, 1997)

Pada Gambar 2.3 menunjukkan *pseudocode* algoritme genetika yang terdiri dari inisialisasi, reproduksi, evaluasi, dan seleksi. Biasanya proses seleksi ini menggunakan tiga metode, yaitu *roulette wheel, binary tournament,* dan *elitism* (Mahmudy, 2013).

1. *Roulette wheel*

Merupakan metode paling sederhana, yaitu menghitung nilai probabilitas setiap individu berdasarkan nilai *fitness*-nya, menghitung prosentase tiap individu dari probabilitas tersebut. Nilai probabilitas tersebut dapat digunakan untuk menghitung nilai probabilitas kumulatif pada proses seleksi setiap individu (Mahmudy, 2015).

2. *Binary tournament*

Menetapkan nilai *tour* pada tiap individu yang dipilih secara *random* dari sebuah populasi. Ukuran *tour* tersebut bernilai 2 (binary). Proses seleksi dilakukan dengan mengambil individu terbaik dalam populasi untuk masuk pada populasi selanjutnya (Mahmudy, 2015).

3. *Elitism*

Semua individu dalam populasi (*parent* dan *child*) dikumpulkan dalam sebuah populasi. Dari populasi tersebut, dipilih individu terbaik sejumlah populasi awal yang dinyatakan lolos dan masuk pada generasi selanjutnya (Mahmudy, 2015).

2.3.4 Parameter Algoritme Genetika

Berikut ini merupakan beberapa parameter dari algoritme genetika (Mahmudy, 2013):

1. Nilai Ukuran Populasi (*popSize*)

   Merupakan jumlah dari kromosom yang terdapat dalam populasi.

2. Jumlah Generasi

   Merupakan nilai dalam algoritme generika yang menyatakan iterasi.

3. *Crossover rate*

Merupakan rasio *child* yang dihasilkan pada proses *crossover* berdasarkan ukuran populasi.

4. *Mutation rate*

Merupakan rasio *child* yang dihasilkan pada proses *mutation* berdasarkan ukuran populasi.

Semakin besar nilai dari masing-masing parameter, maka semakin besar pula solusi terbaik akan dioptimalkan. Tetapi dengan kondisi tersebut akan memakan waktu komputasi yang cukup lama (Mahmudy, 2013).

2.3.5 Kondisi Berhenti atau *Terminator Condition*

Iterasi akan terus dilakukan hingga mencapai kondisi berhenti. Berikut beberapa kriteria yang digunakan untuk kondisi berhenti (Mahmudy, 2013):

1. Iterasi berhenti hingga generasi *n*. Penentuan *n* berdasarkan eksperimen yang telah dilakukan dan nilai tersebut diatur berdasarkan kompleksitas permasalahan yang bertujuan untuk mendapatkan solusi yang terbaik.
2. Iterasi berhenti jika sudah menghasilkan solusi permasalahan yang konvergen.
3. Iterasi berhenti jika mencapai *t* satuan waktu, sehingga kinerja algoritme dapat dibandingkan melalui kondisi ini.

2.4 Simulated Annealing

Algortima *simulated annealing* (SA) pertama kali diperkenalkan oleh Metropolis et al. pada tahun 1953. Algoritme ini diadaptasi dari proses *annealing* pada pembuatan kristal suatu material, yaitu proses pendinginan suatu benda padat sehingga strukturnya membeku pada suatu energi minimal. Pada proses pembuatan kristal suatu material, dilakukan pemanasan hingga mencapai satu

titik tertentu. Dalam keadaan ini, atom-atom akan bergerak bebas dengan tingkat energi yang tinggi. Kemudian, secara perlahan suhu diturunkan dengan harapan energi dapat berkurang menuju ke suatu tingkatan yang relatif rendah. Semakin lambat laju pendinginan, akan semakin rendah pula energi yang dicapai oleh sistem. Dengan demikian, atom-atom tersebut diharapkan akan berada dalam posisi optimum dengan energi yang minimal (Bertsimas & Tsitsiklis, 1998).

Prosedur SA dalam ilmu komputer, dilakukan dengan cara memilih solusi awal yang kemudian dilakukan perubahan struktur pada solusinya. Apabila solusi baru lebih baik, maka solusi baru tersebut dianggap sebagai solusi yang lebih optimal. Dan sebaliknya, apabila solusi baru tidak lebih baik dari sebelumnya maka akan dilakukan pemilihan solusi dengan *probability acceptance*. Karena iterasi awal pada suhu (*Tc*) tinggi maka besar kemungkinan solusi yang lebih buruk terpilih, seperti pada proses *annealing* saat atom bergerak bebas. Hal tersebut akan dilakukan terus sampai kondisi berhenti (Kirkpatrick, et al., 1983).

Dalam penggunaan SA perlu diperhatian empat hal berikut ini (Kirkpatrick, et al., 1983):

1. Konfigurasi sistem yang sederhana (representasi masalah).
2. Memodifikasi kandidat solusi.
3. Fungsi untuk menentukan seberapa optimal suatu kandidat solusi.
4. Kapan parameter *Tc* diturunkan dan seberapa banyak iterasi yang dibutuhkan.

Salah satu metode yang dapat dimodifikasi adalah *neighborhood move*. *Neighborhood move* merupakan prosedur perbaikan sederhana untuk mengoptimalkan kandidat solusi dengan mengubah strukturnya. Perubahan ini bertujuan untuk mengeksplorasi daerah diluar *local optimum* demi tercapainya *global optimum*. Sehingga algoritme ini cocok digunakan untuk operator modifikasi pada algoritme pencarian untuk mengoptimalkan hasil pencarian GA yang terjebak dalam *local optimum*.

Berdasarkan Gambar 2.4, proses *simulated annealing* dimulai dengan memilih satu solusi *Xp*, dilakukan proses *neighborhood move* dengan solusi *Xn*, dan menghitung selisih nilai *fitness* menggunakan Persamaan (2.3) seperti di bawah ini:

$$\Delta f = f(Xn) - f(Xp) \qquad (2.3)$$

Keterangan:
Δf = Selisih nilai *fitness*

$f(Xn)$ = Nilai *fitness* solusi *Xn*

Xn = Individu hasil *neighborhood move*

$f(Xp)$ = Nilai *fitness* solusi *Xp*

Xp = Individu terbaik hasil algoritme genetika

```
Create initial solution Xp
Calculate f(Xp)
Set Tc, Ta, 0<β<1
While (T(c+1) ≤ Ta)
     Select a neighboring solution Xn
     Calculate f(Xn)
     Compute Δf = f(Xn)-f(Xp)
     If (Δf < 0)
          Then Xp=Xn
               Else
                    If 1/(1+e^(-Δf/Tc)) > random (0,1)
                      Then Xp=Xn
                           Else
                                T(c+1) = β.Tc
Until termination condition is satidfied
```

Gambar 2.4 ***Pseudocode Simulated Annealing***

Sumber : (Elhaddad, 2012)

Apabila selisih nilai *fitness* kurang dari nol *(Δf < 0)* maka solusi *Xn* menggantikan solusi *Xp*, jika tidak maka menggunakan pembanding nilai *random* pada Persamaan (2.4) seperti dibawah ini:

$$pRand = \frac{1}{1+e^{\frac{-\Delta_f}{Tc}}} \quad (2.4)$$

Keterangan:

pRand = Pembanding nilai *random*

e = Nilai exponent

Δf = Selisih nilai *fitness*

Tc = Variabel suhu yang menampung $Tc^{(old)}$ pada iterasi sebelumnya dan $Tc^{(new)}$ pada iterasi saat ini

Apabila pembanding nilai *random* lebih besar daripada nilai *random* (0,1) maka solusi *Xn* menggantikan solusi *Xp*, jika tidak maka dilakukan penurunan suhu menggunakan Persamaan (2.5) seperti dibawah ini:

$$Tc^{(new)} = \beta.Tc^{(old)} \quad (2.5)$$

Keterangan:

$Tc^{(new)}$ = Nilai suhu awal setelah diturunkan

β = Koefisien penurunan suhu

$Tc^{(old)}$= Nilai suhu awal sebelum diturunkan saat itu

Proses di atas akan terus dijalankan sampai kondisi penurunan suhu kurang dari sama dengan suhu akhir (*Ta*) terpenuhi.

2.5 Hibridisasi Algoritme Genetika – *Simulated Annealing* (GA-SA)

GA-SA merupakan algoritme pencarian hibridisasi antara algoritme genetika dengan *simulated annealing*. Penggabungan ini bertujuan untuk memanfaatkan kelebihan dari masing-masing algoritme dan saling menutupi kekurangan masing-masing algoritme untuk mendapatkan hasil solusi yang lebih optimal (Al-Milli, 2011). Seperti pada algoritme GA yang mempunyai keefektifan yang tinggi pada saat pencarian solusi, tetapi algoritme SA tidak efisien dalam pencarian karena membutuhkan waktu yang lama.

GA-SA ini tidak berdiri sendiri, melainkan dibantu oleh *neighborhood move*. *Neighborhood move* akan digunakan sebagai operator modifikasi pada siklus *simulated annealing*. Pada siklus *simuated annealing* ini akan diterapkan setelah proses seleksi GA (Norozi, et al., 2011). Pada Gambar 2.5 akan memaparkan alur algoritme GA-SA secara umum.

```
Initialize GA-SA population
While (iterasi < maxIterasi) do
     Select 2 parents from population randomly
     Create child solution using one cut point crossover
     Select parent from population randomly
     Create child solution using reciprocal exchange
     mutation
     Evaluate fitness of all solutions and sort by fitness
     Select population by fitness
     Select the best fitness in population for SA process
     While (T(c+1) ≤ Ta)
     Do neighborhood move process
     Evaluate fitness
     If (improvement)
          Back to neighborhood process
          Else
               Decrease Tc ← T(c+1)
     Until T(c+1) ≤ Ta is satisfied
     Get best results of SA process
End while until iterasi < maxIterasi
The best solution achieved as output
```

Gambar 2.5 Pseudocode Hibridisasi GA-SA

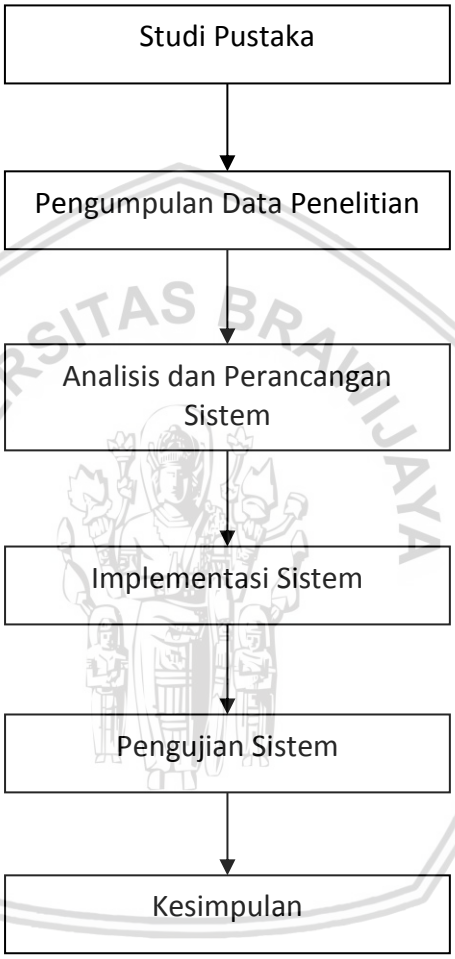
Langkah-langkah GA-SA menurut (Norozi, et al., 2011) berdasarkan Gambar 2.4 adalah sebagai berikut:

1. Representasi kromosom secara *random*.
2. Menerapkan *crossover* pada kromosom yang telah dipilih untuk mendapatkan individu baru *(child)*.
3. Menerapkan operator mutasi untuk mendapatkan kromosom baru.
4. Evaluasi kromosom baru.
5. Melakukan proses seleksi.

6. Apabila kondisi berhenti terpenuhi maka selesai, jika tidak akan dijalankan algoritme *simulated annealing* dengan metode *neighborhood move*.
7. Setelah itu mengevaluasi nilai *fitness*.
8. Apabila terjadi perbaikan, maka memproses metode *neighborhood move* lagi. Jika tidak maka dilakukan penurunan *temperature*.
9. Apabila $T(c+1) \leq Ta$ terpenuhi maka akan dilakukan proses seleksi terhadap individu baru tersebut dengan individu hasil *algoritme* genetika untuk mendapatkan solusi terbaik.
10. Selanjutnya mengulang proses ke-2 sampai dengan 9 sampai kondisi berhenti terpenuhi.

BAB 3 METODOLOGI

Pada bab ini menjelaskan langkah-langkah dalam metodologi penelitian yang terdiri dari tahapan studi pustaka, pengumpulan data penelitian, analisis dan perancangan sistem, implementasi sistem, pengujian sistem, serta kesimpulan. Berikut adalah tahapan metodologi penelitian yang ditunjukkan pada Gambar 3.1.



Gambar 3.1 Blok Diagram Metode Penelitian

3.1 Studi Pustaka

Pada langkah pertama ini, yaitu studi pustaka dilakukan untuk mempelajari dan memahami secara mendalam mengenai teori-teori dasar keilmuan yang merupakan bahan dasar untuk penelitian yang akan dilakukan. Teori-teori dasar keilmuan tersebut dapat berasal dari pakar-pakar, buku, jurnal, *e-book*, penelitian sebelumnya, dan dari sumber pustaka lain yang masih berkaitan dengan peneliatian ini serta dapat dipertanggung jawabkan. Dan teori-teori tersebut meliputi:

1. Pengetahuan tentang penjadwalan mata pelajaran.

Pengetahuan mengenai bagaimana penjadwalan mata pelajaran yang terdapat di SMA Negeri 6 Surabaya.

2. Pengetahuan tentang algoritme genetika.

Pengetahuan mengenai bagaimana sebuah algoritme genetika dapat digunakan untuk komputasi penyelesaian masalah.

3. Pengetahuan tentang *simulated annealing*.

Pengetahuan mengenai bagaimana sebuah *simulated annealing* dapat dikombinasikan dengan algoritme genetika untuk mengahasilkan hasil optimal yang lebih baik.

3.2 Pengumpulan Data Penelitian

Data yang dikumpulkan dalam penelitian ini adalah data-data yang menunjang dalam penjadwalan mata pelajaran di SMA Negeri 6 Surabaya, yang mana data yang dikumpulkan didapat dari proses wawancara kepada pihak kurikulum SMA Negeri 6 Surabaya, Dra. Ika Mustikawati, M.pd beserta *survey* langsung ke lokasi untuk mengetahui bagaimana proses penjadwalan sebelumnya pada bulan Oktober-November 2017.

Kebutuhan yang dibutuhkan antara lain jumlah ruang kelas sejumlah 24 ruangan, jumlah jam mengajar dalam satu minggu sejumlah 45 jam, dan juga total jam mata pelajaran pada semua guru berjumlah 1036. Dapat dilihat hasil perkalian antara jumlah ruangan dengan jumlah jam mengajar dalam satu minggu lebih besar daripada total jam mata pelajaran pada setiap guru, hal ini dapat membuktikan bahwa penjadwalan mata pelajaran pada studi kasus SMA Negeri 6 Surabaya dapat dijadwalkan.

3.3 Analisis dan Perancangan Sistem

Analisis digunakan untuk mengetahui kebutuhan-kebutuhan apa saja yang diperlukan dalam membangun sebuah sistem penjadwalan mata pelajaran pada studi kasus SMA Negeri 6 Surabaya. Sedangkan untuk perancangan sistem dibutuhkan sebagai rancangan langkah kerja dalam membangun sebuah sistem penjadwalan mata pelajaran pada studi kasus SMA Negeri 6 Surabaya, baik dari segi model ataupun arsitektur untuk mempermudah pengimplementasian.

3.3.1 Analisis Kebutuhan

Pada tahap ini dilakukan penggalian kebutuhan-kebutuhan yang diperlukan dalam membangun sebuah sistem penjadwalan mata pelajaran pada studi kasus SMA Negeri 6 Surabaya, baik dari kebutuhan *hardware*, kebutuhan *software*, maupun kebutuhan data.

1. *Hardware* yang digunakan:
    a. RAM 4.00 GB

b. Hardisk dengan kapasitas sebesar 500GB

c. Monitor 14 Inch

2. *Software* yang digunakan:

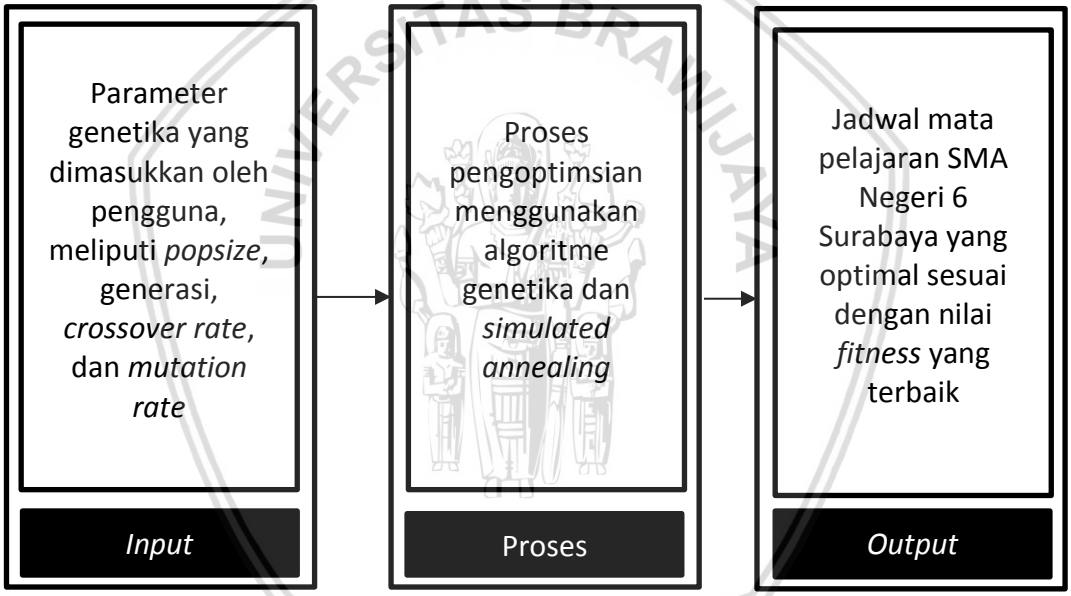
d. Sistem operasi windows 8

e. NetBeans IDE 8.2

f. Java SE Development Kit 8 Update 60 (64-bit)

3.3.2 Perancangan Sistem

Selanjutnya pada tahap perancangan sistem berisi tentang bagaimana rancangan langkah-langkah dalam membangun sebuah proses penjadwalan mata pelajaran pada studi kasus SMA Negeri 6 Surabaya dengan menerapkan algoritme genetika dan *simulated annealing* dalam proses komputasinya. Dapat dilihat pada Gambar 3.2.



Gambar 3.2 Blok Diagram Perancangan Sistem

3.4 Implementasi sistem

Pada tahap ini dilakukan implementasi perangkat lunak dengan menerapkan algoritme genetika dan *simulated annealing* dalam optimasi penjadwalan mata pelajaran di SMA Negeri 6 Surabaya. Implementasi tersebut meliputi:

1. Membuat interface untuk pengguna, dengan memasukkan parameter algoritme genetika ke dalam *form* yang tersedia. Pengimplementasian tersebut menggunakan *software* NetBeans dengan bahasa pemrograman Java.

2. Pengimplementasian perhitungan dengan hibridisasi algoritme genetika dan *simulated annealing* dalam sistem optimasi penjadwalan mata pelajaran di SMA Negeri 6 Surabaya ke dalam bahasa pe*mr*ograman Java.

3.5 Pengujian Sistem

Pengujian sistem dilakukan untuk menguji seberapa optimal hasil rancangan solusi sistem dengan menerapkan algoritme genetika dan *simulated annealing* dengan beberapa perlakuan berbeda. Misalnya perubahan terhadap parameter baik itu dari segi *popSize*, jumlah generasi, *crossover rate*, *mutation rate*, atau bahkan pengujian terhadap *algoritme*.

3.6 Kesimpulan

Tahap terakhir ini dilakukan setelah semua tahapan sebelumnya telah diselesaikan seperti perancangan, implementasi, dan pengujian sistem. Kesimpulan ini diambil dari hasil pengujian dan analisis yang telah dilakukan sebelumnya. Disamping itu pula, kesimpulan merupakan jawaban dari rumusan masalah yang sebelumnya telah dibuat dan dapat berisi saran untuk penelitian selanjutnya.

BAB 4 PERANCANGAN

Pada bab perancangan ini akan membahas mengenai formulasi permasalahan, siklus penyelesaian masalah, perhitungan manual atau manualisasi, perancangan pengujian, serta perancangan antarmuka sistem dalam optimasi penjadwalan mata pelajaran kurikulum 2013 dengan menggunakan hibridisasi algoritme genetika dan *simulated annealing* pada studi kasus SMA Negeri 6 Surabaya.

4.1 Formulasi Permasalahan

Dalam penelitian ini masalah yang akan diselesaikan adalah masalah yang terkait dengan penjadwalan mata pelajaran yang ada di SMA Negeri 6 Surabaya. Sebelumnya dalam penjadwalan mata pelajaran di SMA Negeri 6 Surabaya masih menggunakan cara yang sederhana, yaitu dikerjakan oleh bagian waka kurikulum dibantu *staff* dengan menggunakan *tools* Microsoft Excel. Dalam penjadwalan mata pelajaran dibutuhkan beberapa data, seperti data ruang kelas, data guru, data mata pelajaran serta data pembagian tugas mengajar guru. Dari data-data tersebut akan dibentuk jadwal mata pelajaran. Penjadwalan yang terbentuk tidak hanya penjadwalan yang tidak memiliki bentrok jam atau guru saja, tetapi penjadwalan tersebut dapat memenuhi beberapa *constraint* yang telah didefinisikan. *Constraint* penjadwalan mata pelajaran di SMA Negeri 6 Surabaya dapat dilihat pada Tabel 4.1

Tabel 4.1 *Constraint* Penjadwalan Mata Pelajaran SMA Negeri 6 Surabaya

NO.	*CONSTRAINT*
1	Guru yang mengajar tidak boleh mengajar lebih dari satu kelas atau mengajar di lain kelas pada waktu yang sama (bentrok)
2	Mata pelajaran Penjaskes OR hanya boleh berada pada jam pelajaran ke 1-4
3	Guru yang mengajar tidak boleh mengajar lebih dari empat jam pelajaran dalam mata pelajaran yang sama dan di hari yang sama
4	Dalam satu mata pelajaran yang sama pada jam kedua dan selanjutnya (sejumlah ketentuan jumlah jamnya) tidak boleh beda hari

Dalam penelitian ini akan dibangun sebuah sistem yang dapat mengoptimasi penjadwalan mata pelajaran dengan menggunakan hibridisasi algoritme genetika dan *simulated annealing*. Serta dapat menghasilkan penjadwalan yang sesuai dengan *constraint* yang telah didefiniskan.

Data masukan yang dibutuhkan oleh sistem berupa data ruang kelas, data guru, data jam pelajaran, data mata pelajaran serta data penugasan guru mengajar. Untuk data pembagian tugas mengajar guru perlu dilakukan proses *encoding* terlebih dahulu agar data asli dapat diproses oleh algoritme genetika

dan *simulated annealing*. Proses *encoding* dilakukan untuk membuat data dengan kode unik yang dapat mewakili seluruh data penugasan. Setelah itu dilakukan proses representasi kromsom, sebagai salah satu elemen yang ada pada algoritme genetika. Selain data-data yang telah disebutkan sebelumnya, sistem juga memerlukan beberapa masukan berupa parameter-parameter algoritme yang terdiri dari jumlah populasi, jumlah generasi (iterasi maksimal), nilai *crossover rate* (*cr*), nilai *mutation rate* (*mr*), suhu saat ini (*Tc*), suhu akhir (*Ta*), dan koefisien penurunan suhu (*β*). Data yang dihasilkan dari proses *encoding* akan dilakukan pencarian solusi dengan menggunakan algoritme genetika dan *simulated annealing* sesuai dengan parameter yang telah dimasukkan. Solusi yang dihasilkan dari proses tersebut kemudian dilakukan proses *decoding* untuk pengaplikasian solusi menjadi bentuk jadwal mata pelajaran yang akan digunakan dalam proses pembelajaran di SMA Negeri 6 Surabaya. Data yang digunakan dalam penjadwalan ini adalah data ruang kelas pada Tabel 4.2, data guru pada Tabel 4.3, serta data pembagian tugas mengajar tahun ajaran 2017/2018 pada Tabel 4.4.

Tabel 4.2 Data Ruang Kelas

Kode Ruang	Kelas
0	X IPA 1
1	X IPA 2
2	X IPA 3
3	X IPA 4
4	X IPA 5
5	X IPA 6
6	X IPS 1
7	X IPS 2
8	XI IPA 1
9	XI IPA 2
10	XI IPA 3
11	XI IPA 4
12	XI IPA 5
13	XI IPA 6
14	XI IPS 1
15	XI IPS 2
16	XII IPA 1
17	XII IPA 2
18	XII IPA 3
19	XII IPA 4
20	XII IPA 5
21	XII IPA 6
22	XII IPS 1
23	XII IPS 2

Tabel 4.3 Data Guru

KODE GURU	NAMA GURU
0	Dra. Hj. Soetji Sri Handajani
1	Hj. Sri Wachjuni, M.Pd
2	Dra. Ika Mustikawati, M.Pd
3	Dra. Hj. Suryati
4	Dra. Endah Herawati N, SS., M.Pd
5	Endang Megawati, S.Pd., M.Pd.I
6	Dra. RR. Dyah Retnowati
7	Drs. Samsul Arifin
8	Drs. H. Herri Zumidar Halim, M.Pd
9	Sri Handayani, S.Pd
10	Drs. Hari Indarjoko, M.Pd
11	Arni Wiyati, S.Pd
12	Sri Utami, M.Si
13	Drs. H. Yatimun
14	Ummu Hani, SS
15	Dian Ariani, S.Pd
16	Munif Musthofa, SPd.I
17	Muhammad Rendik Widiyanto, S.Pd
18	Azward Annas Ro'is Habibi, S.Pd
19	Yeni Hidayati, S.Pd
20	Mochamad Sobani, S.Pd
21	Dra. Enny Sjofinar Joesoef
22	Winarni, M.Pd
23	Drs. H. Hari Sutanto, M.Pd
24	Drs. Aries Prasetyo
25	Irfa Rochimah Alfi, S.Si, M.Pd.
26	Ety Yuniandari, S.Pd.
27	Endah Suryani, S.Pd
28	Misriyanto, S.Pd
29	Sri Bintang Ratnaningsih, S.Si., M.Pd
30	Ahmad Musyafiq, S.Pd, M.Si
31	Agus Widiyono, S.Pd
32	Sri Suratni, S.Pd
33	Evie Rakhmalia, S.Pd., Gr
34	Drs. Widianto Riyadi
35	Dra. Cicik Kurniasih
36	Drs. Soleman
37	Hj. Rr. Emmy Padmi W, S.Pd
38	Dra. Sri Wahjuni
39	Drs. H. Djoko Prasitimtoro
40	Yuty Rahinawati
41	Emmilliana Sri Setiti, S.Pd
42	Drs. *i* Ketut Artha
43	Lukas Yanuar H, STh

Tabel 4.4 Data Pembagian Tugas Mngajar

SK PEMBAGIAN TUGAS MENGAJAR							
KODE GURU	**NAMA GURU**	**MATA PELAJARAN**	**KELAS**	**JUMLAH ROMBEL**	**JUMLAH JAM/MINGGU**	**JUMLAH JAM**	**TOTAL JAM**
0	Dra. Hj. Soetji Sri Handajani	Penjaskes OR	X IPA 1, 2, 3, 4, 5, 6	6	3	18	36
			XII IPA 1, 2, 3, 4	4	3	12	

			X IPS 1, 2	2	3	6	
1	Hj. Sri Wachjuni, M.Pd	Matematika Umum	XII IPA 1, 2, 3, 4, 5, 6	6	4	24	24
2	Dra. Ika Mustikawati, M.Pd	Biologi	XI IPA 3	1	4	4	24
			XII IPA 1, 3, 4, 6	4	4	16	
		PKWU	X IPA 2, 3	2	2	4	
…	…	…	…	…	…	…	…
39	Drs. H. Djoko Prasitimtoro	Penjaskes OR	XI IPA 1, 2, 3, 4, 5, 6	6	3	18	36
			XII IPA 5, 6	2	3	6	
			XI IPS 1, 2	2	3	6	
			XII IPS 1, 2	2	3	6	
40	Yuty Rahinawati	Kimia	XI IPA 3, 4, 5	3	4	12	24
			XII IPA 1, 2, 4	3	4	12	

Dari data pembagian tugas mengajar pada Tabel 4.4 dilakukan proses *encoding*, agar dapat digunakan sebagai representasi kromosom untuk menjalankan proses siklus algoritme genetika dalam pencarian solusi penjadwalan. Tabel 4.4 selengkapnya dapat dilihat pada halaman lampiran. Contoh hasil *encoding* dapat dilihat pada Tabel 4.5.

Tabel 4.5 Pengkodean Penugasan Mengajar

DATA KODE TUGAS MENGAJAR							
NO.	**KODE GURU**	**NAMA GURU**	**KODE MATA PELAJARAN**	**MATA PELAJARAN**	**KODE KELAS**	**KELAS**	**JUMLAH JAM/MINGGU**
1	0	Dra. Hj. Soetji Sri Handajani	0	Penjaskes OR	0	X IPA 1	3
2	0	Dra. Hj. Soetji Sri Handajani	0	Penjaskes OR	1	X IPA 2	3
3	0	Dra. Hj. Soetji Sri Handajani	0	Penjaskes OR	2	X IPA 3	3
4	0	Dra. Hj. Soetji Sri Handajani	0	Penjaskes OR	3	X IPA 4	3
5	0	Dra. Hj. Soetji Sri Handajani	0	Penjaskes OR	4	X IPA 5	3
…	…	…	…	…	…	…	…
336	40	Yuty Rahinawati	10	Kimia	11	XI IPA 4	4
337	40	Yuty Rahinawati	10	Kimia	12	XI IPA 5	4
338	40	Yuty Rahinawati	10	Kimia	16	XII IPA 1	4
339	40	Yuty Rahinawati	10	Kimia	17	XII IPA 2	4
340	40	Yuty Rahinawati	10	Kimia	19	XII IPA 4	4

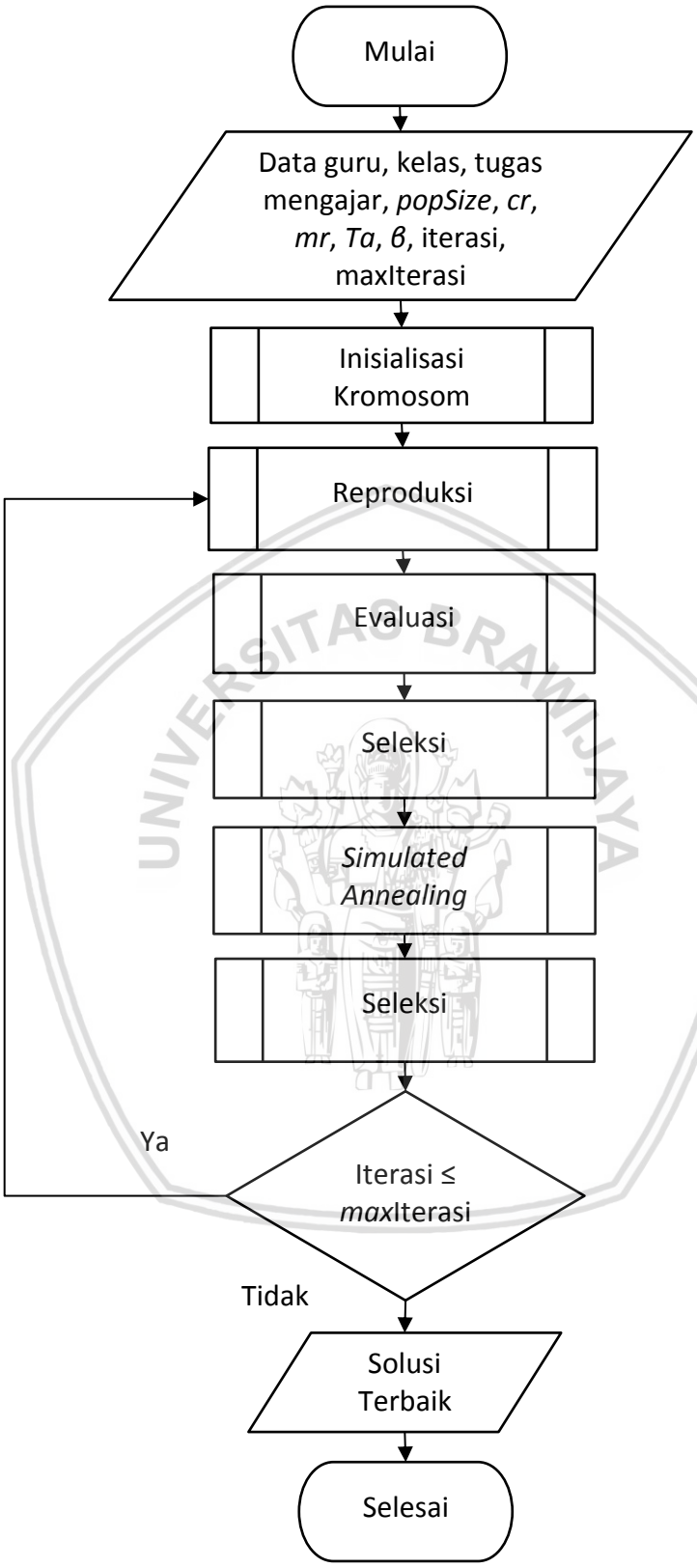
Dari Tabel 4.5 akan digunakan sebagai representasi kromosom, yang mana setiap kode akan mengisi satu gen dari suatu susunan kromosom. Karena gen merupakan representasi dari kode penugasan mengajar, maka representasi panjang kromosom didapat dari banyaknya jumlah data dari pengkodean penugasan mengajar. Tabel 4.5 selengkapnya dapat dilihat pada halaman lampiran.

4.2 Siklus Hibridisasi Algoritme Genetika – *Simulated Annealing* (GA-SA)

Pada siklus ini menjelaskan tentang alur penyelesaian masalah terkait penjadwalan mata pelajaran dengan menggunakan hibridisasi algoritme genetika dan *simulated annealing* (GA-SA). Langkah pertama dilakukan dengan memasukkan data-data apa saja yang dibutuhkan seperti data guru, data kelas, data mata pelajaran, dan data pembagian tugas mengajar dalam bentuk *note* yang berisi kode guru, kode kelas, kode mata pelajaran, dan kode pembagian tugas mengajar. Selain itu dibutuhkan beberapa parameter terkait dengan hibridisasi algoritme GA-SA yaitu jumlah generasi (iterasi maksimal), nilai *crossover rate* (*cr*), nilai *mutation rate* (*mr*), suhu saat ini (*Tc*), suhu akhir (*Ta*), dan koefisien penurunan suhu (*β*).

Pada proses algoritme genetika dimulai dengan inisialisasi kromosom setiap individu di awal populasi. Selanjutnya dilakukan proses reproduksi, yang mana dalam proses ini terdapat dua tahap yaitu *crossover* dan *mutation*. Proses *crossover* dilakukan sampai mendapat sejumlah *cr* x *popSize* individu (*child*), metode *crossover* yang digunakan adalah *one cut-point crossover*. Selain itu proses *mutation* dilakukan sampai mendapat sejumlah *mr* x *popSize* individu (*child*), metode *mutation* yang digunakan adalah *reciprocal exchange mutation*. Proses reproduksi ini dilakukan untuk meningkatkan keberagaman individu sehingga ruang pencariannya lebih luas. Selanjutnya individu-individu tersebut di evaluasi dengan melakukan perhitungan nilai *fitness* pada tiap kromosom dan di *sorting* berdasarkan nilai *fitness* paling besar. Proses terakhir adalah seleksi, dengan mengambil sejumlah *popSize* individu dengan nilai *fitness* terbaik.

Lalu pada proses *simulated annealing* diawali dengan mengambil satu individu terbaik dari hasil algoritme genetika. Variabel parameter yang digunakan dalam proses *simulated annealing* adalah suhu saat ini (*Tc*), suhu akhir (*Ta*), dan koefisien penurunan suhu (*β*). Selanjutnya akan dilakukan modifikasi dengan menggunakan *neighborhood move* terhadap individu yang akan diproses (*P*). *Neighborhood move* merupakan proses penukaran dua nilai gen yang berbeda untuk mendapatkan individu baru (*C*), setelah itu dilakukan evaluasi untuk mendapakan nilai *fitness* dari individu *C*. Nilai fitness dari kedua individu telah didapat, selanjutnya dihitung selisih nilai fitness seperti pada Persamaan (2.3). Apabila selisih nilai *fitness* kurang dari nol *(Δf < 0)* maka individu *C* menggantikan individu *P*, jika tidak maka menggunakan pembanding nilai *random* seperti pada Persamaan (2.4) Apabila pembanding nilai *random* lebih besar daripada nilai *random* (0,1) maka individu *C* menggantikan individu *P*, jika tidak maka dilakukan penurunan suhu menggunakan Persamaan (2.5). Apabila penurunan suhu belum terpenuhi maka harus kembali ke proses *simulated annealing* sampai penurunan suhu terpenuhi. Siklus hibridisasi GA-SA dapat dilihat pada Gambar 4.1.



Gambar 4.1 Siklus Hibridisasi GA-SA

Proses yang dilakukan dalam Siklus Hibridisasi GA-SA pada Gambar 4.1 antara lain:

1. Sistem menerima masukan berupa data guru, data kelas, data tugas mengajar, *popSize, crossover rate* (*cr*), *mutation rate* (*mr*), suhu saat ini (*Tc*), suhu akhir (*Ta*), dan koefisien penurunan suhu (*β*), iterasi, dan maxIterasi.
2. Melakukan proses inisialisasi kromosom untuk membentuk satu individu sebagai representasi solusi.
3. Selanjutnya dilakukan proses reproduksi untuk menghasilkan *child* atau individu baru.
4. Setelah proses reproduksi selesai, dilakukan proses evaluasi dengan menghitung nilai *fitness* dari masing-masing individu.
5. Nilai *fitness* dari masing-masing kromosom telah didapat, maka akan diseleksi dengan mengambil individu nilai *fitness* tertinggi sebanyak jumlah populasi awal.
6. Selanjutnya individu terbaik dari proses algoritme genetika masuk pada proses *simulated annealing*.
7. Individu terbaik dari proses *simulated annealing* akan diseleksi lagi untuk mengambil indivdu terbaik dari hibridisasi GA-SA.
8. Apabila proses hibridisasi GA-SA sudah sama dengan atau lebih dari maxIterasi maka satu individu terbaik dari proses seleksi sebelumnya merupakan representasi solusi terbaik.
9. Apabila proses hibridisasi GA-SA kurang dari sama dengan maxIterasi maka akan kembali pada proses reproduksi.

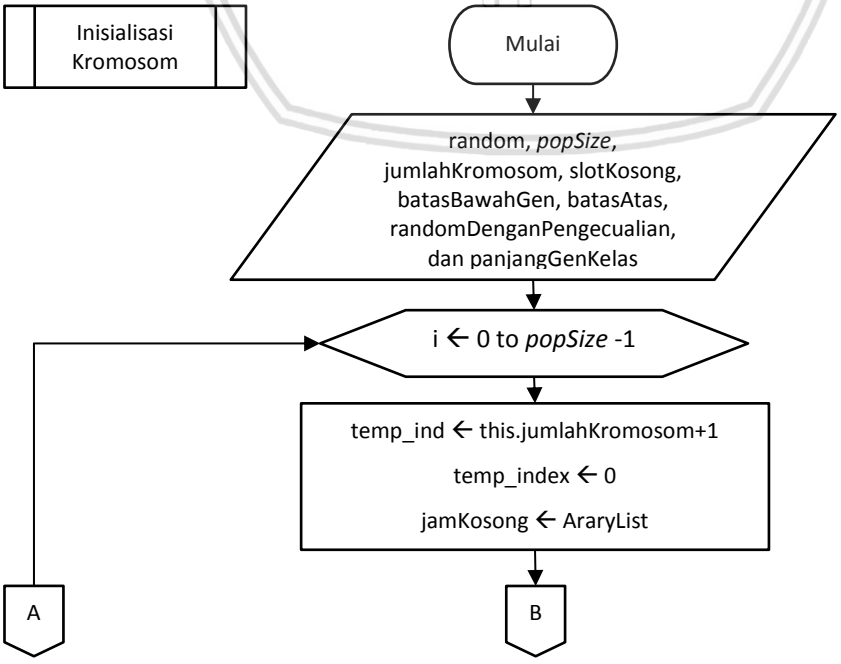
4.2.2 Inisialisasi Kromosom

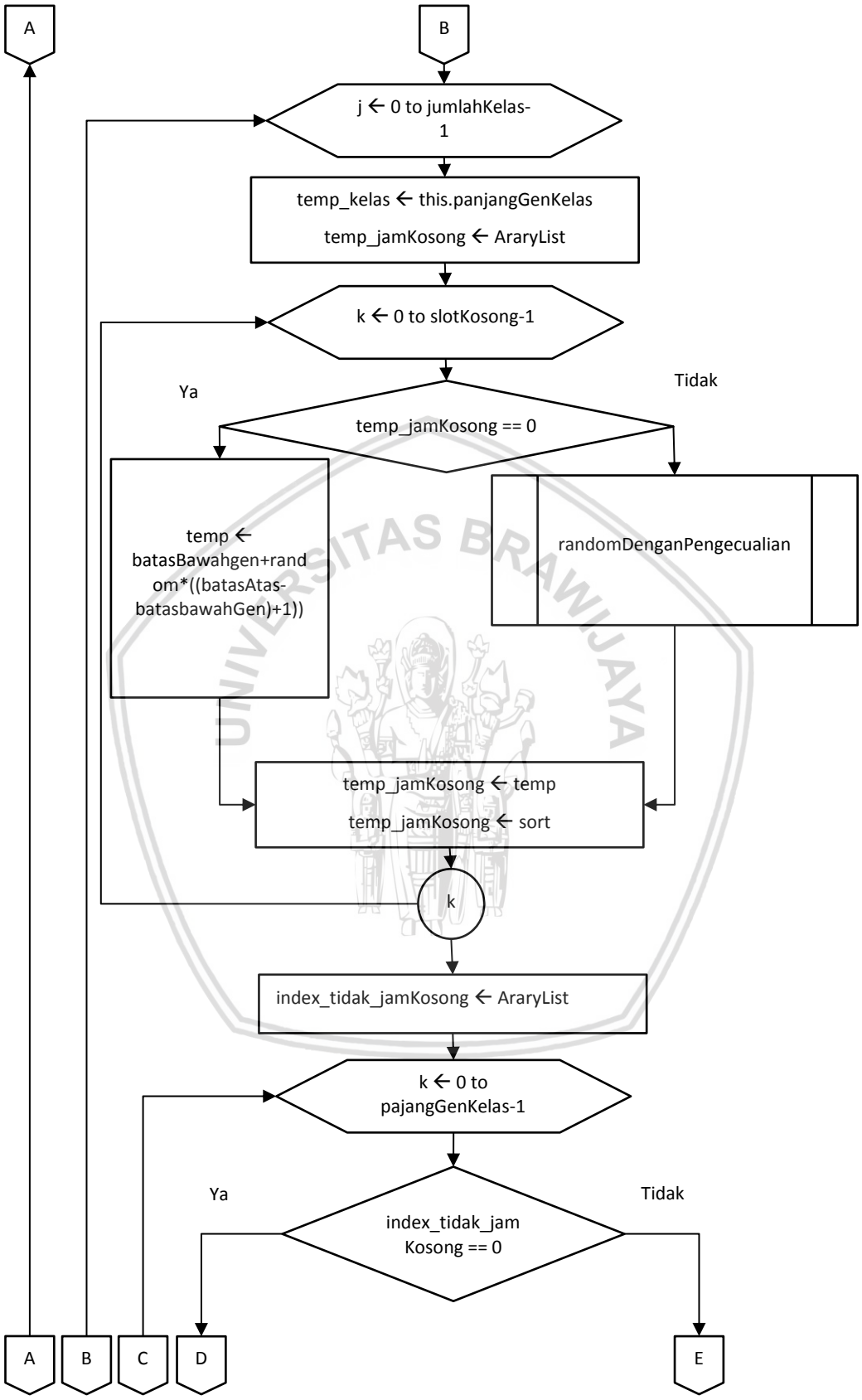
Proses inisialisasi kromosom diperoleh dengan membangkitkan nilai dari data kode tugas mengajar. Panjang kromosom didapat dari jumlah keseluruhan hasil pengurangan antara jumlah jam pelajaran dari hari senin-selasa dengan total jam pada masing-masing tingkatan kelas (X,XI,XII) dari hari senin-selasa dan ditambah dengan slot masing-masing tingkatan kelas (X,XI,XII). Kromosom yang diinisialisasi akan digunakan untuk membangkitkan populasi awal, yang mana setiap kromosom yang diinisialisasi merupakan representasi dari solusi permasalahan. Blok diagram inisialisasi kromosom dapat dilihat pada Gambar 4.2

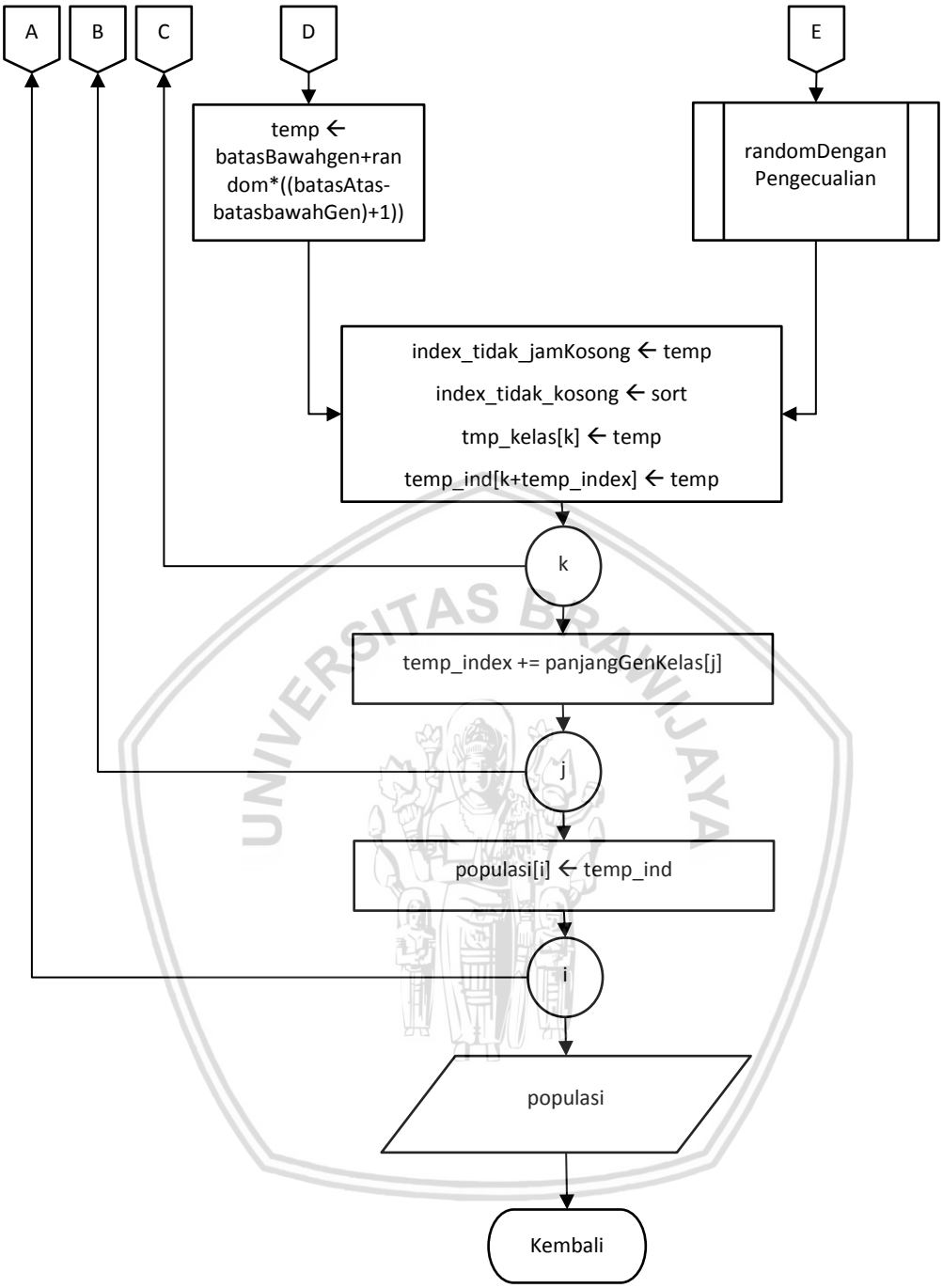
Proses yang dilakukan dalam inisialisasi kromosom pada Gambar 4.2 antara lain:

1. Sistem menerima masukan berupa nilai *random*, *popSize*, jumlahKromosom, slotKosong, batasBawahGen, batasAtas, *random*DenganPengecualian, dan panjangGenKelas.

2. Perulangan variabel *i* sebanyak kurang dari jumlah *popSize*, selanjutkan dilakukan inisialisasi variabel temp_ind, temp_index, dan jamKosong.
3. Perulangan variabel *j* sebanyak kurang dari jumlahKelas, selanjutnya dilakukan inisialisasi variabel temp_kelas, dan temp_jamKosong.
4. Perulangan variabel *k* sebanyak kurang dari jumlah slotKosong, selanjutnya dilakukan deklarasi variabel temp. Apabila variabel temp_jamKosong kosong maka variabel akan diinisialisasi ulang, jika tidak maka akan memanggil method *random*DenganPengecualian.
5. Setelah itu nilai dari variabel temp akan ditambahkan ke dalam variabel temp_jamKosong dan disorting.
6. Perulangan variabel *k* sebanyak kurang dari jumlah panjangGenKelas, selanjutnya apabila temp_jamKosong tidak mengandung variabel *k* dilakukan deklarasi variabel temp. Apabila variabel index_tidak_jamKosong kosong maka variabel akan diinisialisasi ulang, jika tidak maka akan memanggil method *random*DenganPengecualian.
7. Setelah itu nilai dari variabel temp akan ditambahkan ke dalam variabel index_tidak_jamKosong dan disorting. Nilai dari variabel temp_kelas[*k*] dan temp_ind[*k*+temp_index] akan menggantikan nilai dari variabel temp.
8. Apabila perulangan variabel *k* sebanyak kurang dari jumlah panjangGenKelas terpenuhi, maka akan perbaikan nilai pada variabel temp_index.
9. Apabila perulangan variabel *j* sebanyak kurang dari jumlahKelas terpenuhi, maka nilai populasi akan bertambah.
10. Nilai populasi akan terus bertambah sebanyak *popSize*, setelah terpenuhi maka akan dikembalikan nilai dari populasi tersebut dan inisialisasi kromosom berhasil dilakukan.




A
B
j ← 0 to jumlahKelas-1
temp_kelas ← this.panjangGenKelas
temp_jamKosong ← ArararyList
k ← 0 to slotKosong-1
Ya
Tidak
temp_jamKosong == 0
temp ← batasBawahgen+random*((batasAtas-batasbawahGen)+1))
randomDenganPengecualian
temp_jamKosong ← temp
temp_jamKosong ← sort
k
index_tidak_jamKosong ← AraryList
k ← 0 to pajangGenKelas-1
Ya
Tidak
index_tidak_jam Kosong == 0
A
B
C
D
E

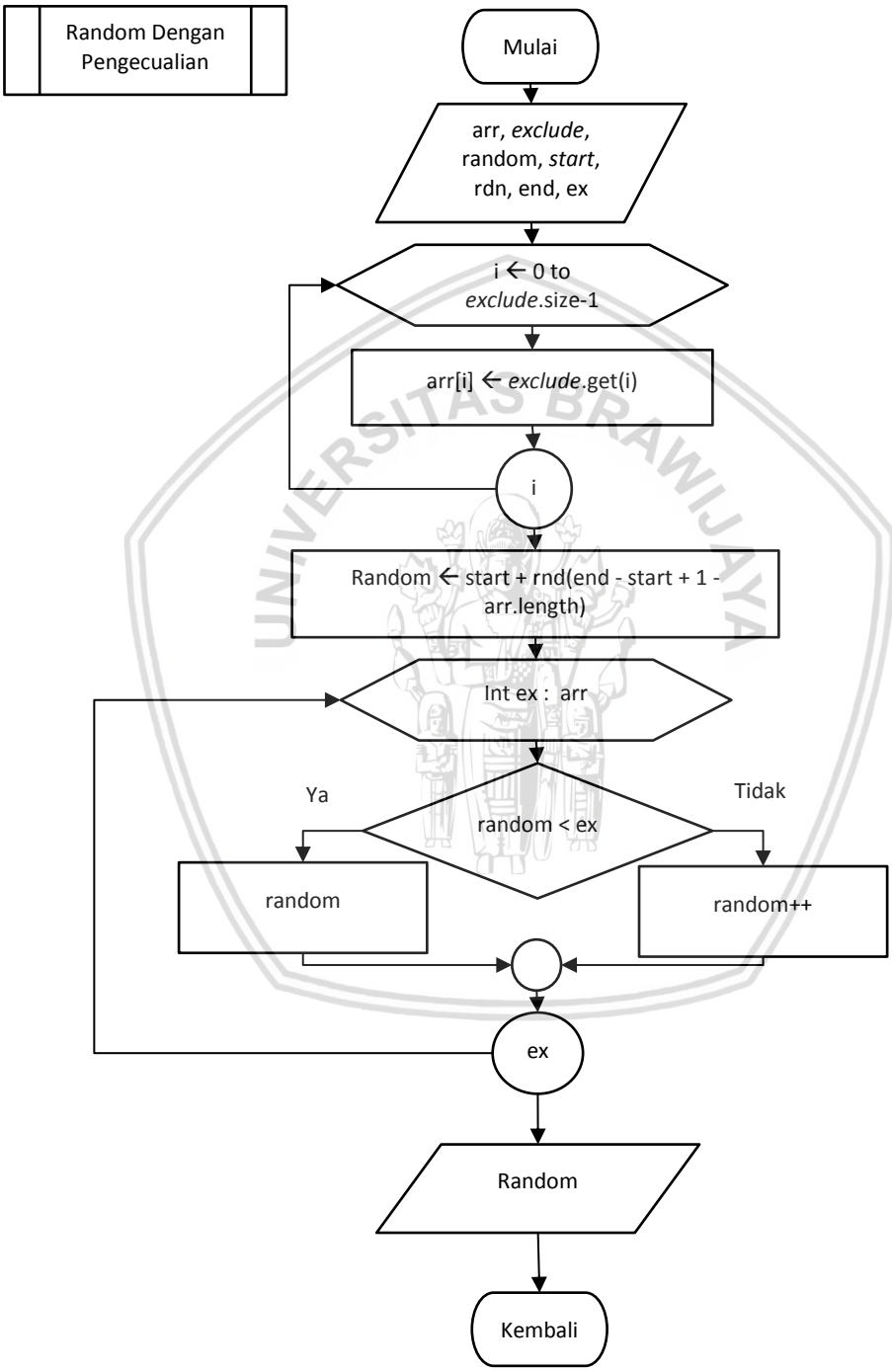


Gambar 4.2 Blok Diagram Inisialisasi Kromosom

Proses yang dilakukan dalam *random* dengan pengecualian pada Gambar 4.3 antara lain:

1. Sistem menerima masukan berupa variabel arr, *exclude*, *random*, *start*, rdn, *end*, dan ex.
2. Inisialisasi variabel arr sebanyak ukuran *exclude*.
3. Perulangan variabel *i* sebanyak kurang dari ukuran *exclude* yang berisi inisialisasi nilai variabel arr dengan index *i*.

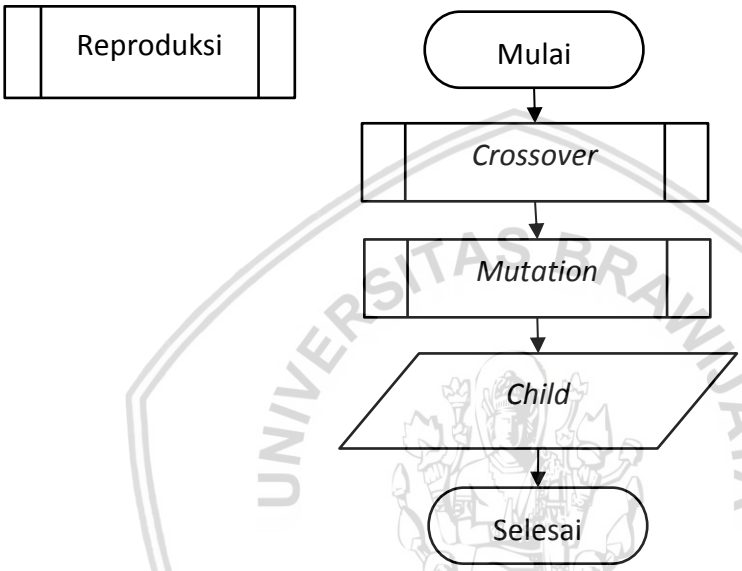
4. Inisialisasi variabel *random*.
5. Perulangan variabel ex, jika nilai *random* kurang dari ex maka perulangan berhenti. Jika tidak, maka jumlah nilai pada variabel *random* bertambah.
6. Pengembalian nilai *random* dan didapatkan hasil *random* dengan pegecualian.



Gambar 4.3 Blok Diagram *Random* Dengan Pengecualian

4.2.3 Reproduksi

Proses reproduksi dilakukan untuk menghasilkan anak (*child*) atau individu baru dalam populasi. Terdapat dua macam cara reproduksi, yaitu dengan cara *crossover* atau perkawinan silang dan cara *mutation* atau mutasi. Proses *crossover* diakukan dengan cara mengkawinkan silang antara dua buah individu (*parent*) dalam populasi. Sedangkan *mutation* hanya membutuhkan satu *parent* dalam proses reproduksinya. Untuk jumlah anak (*child*) ditentukan oleh parameter *cr* (*crossover rate*) dan *mr* (*mutation rate*). Blok diagram proses reproduksi dapat dilihat pada Gambar 4.4.



Gambar 4.4 Blok Diagram Reproduksi

Proses yang dilakukan dalam reproduksi pada Gambar 4.4 antara lain:

1. Sistem melakukan proses *crossover*.
2. Setelah itu sistem melakukan proses *mutation*.
3. Sehingga sistem menghasilkan keluaran berupa *child*.

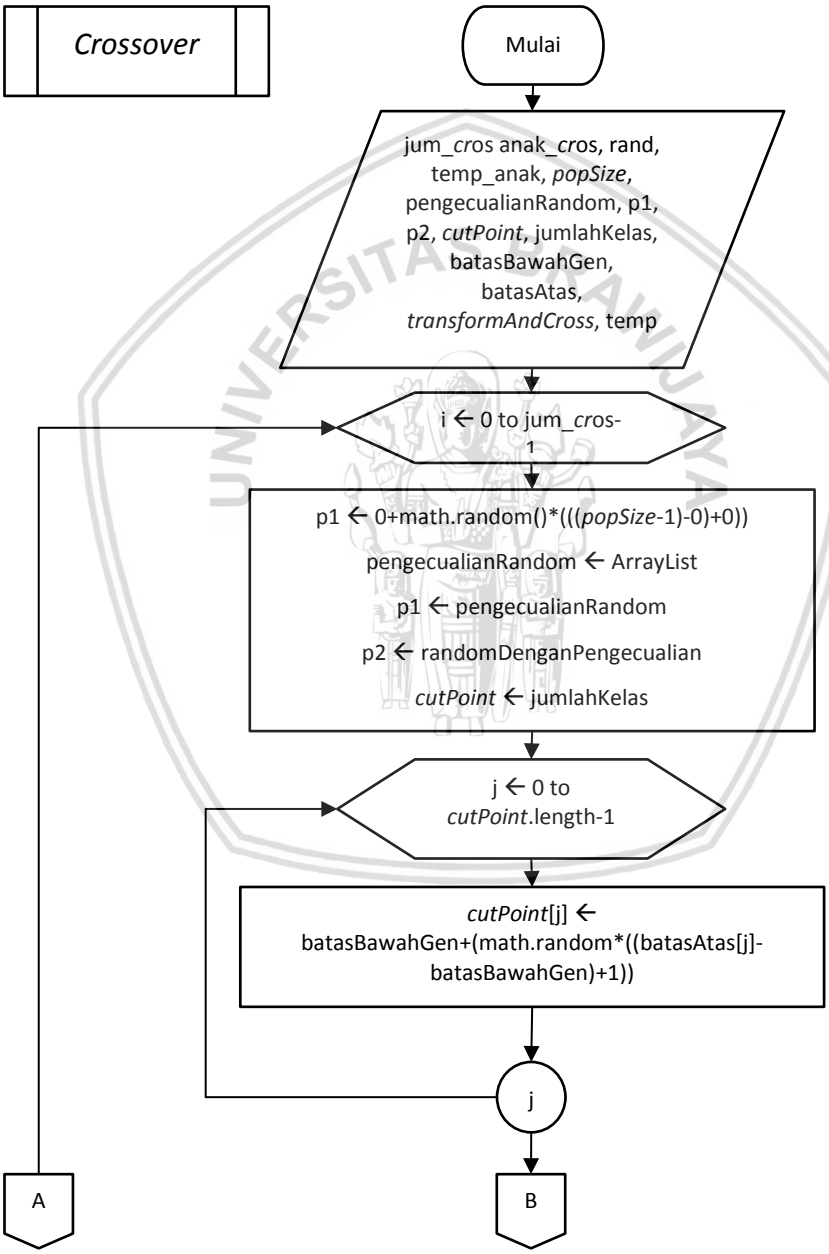
4.2.3.2 Crossover

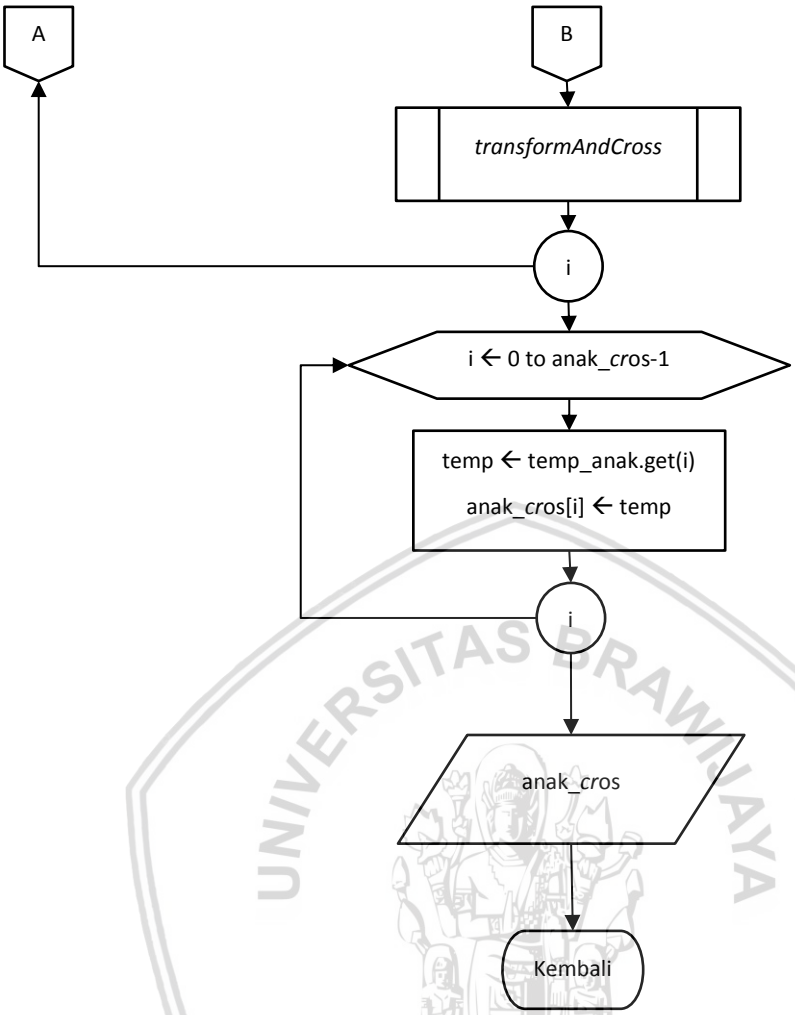
Proses *crossover* merupakan salah satu cara dari proses reproduksi dengan mengkawin silangkan dua buah *parent* untuk menghasilkan *child*. Dalam penelitian ini proses *crossover* yang dilakukan dengan cara *one-cut point,* yaitu menukarkan susunan kromosom denga satu titik potong. Blok diagram *one-cut point crossover* dapat dilihat pada Gambar 4.5.

Proses yang dilakukan dalam *one-cut point crossover* pada Gambar 4.5 antara lain:

1. Sistem menerima masukan berupa jum_*cr*os anak_*cr*os, rand, temp_anak, *popSize*, pengecualian*Random*, p1, p2, *cutPoint*, jumlahKelas, batasBawahGen, batasAtas, *transformAndCross*, temp.

2. Perulangan variabel *i* sebanyak kurang dari jum_*cr*os yang berisi inisialisasi variabel p1, p2, dan *cutPoint* .
3. Perulangan variabel *j* sebanyak kurang dari panjang *cutPoint* yang berisi inisialisasi variabel *cutPoint.*
4. Memasukkan variabel temp_anak ke dalam method *transformAndCross*.
5. Perulangan varaibel *i* sebanyak kurang dari anak_*cr*os yang berisi inisialisasi variabel temp dan anak_*cr*os.
6. Pengembalian nilai anak_*cr*os dan menghasilkan *child* dari proses *crossover*.



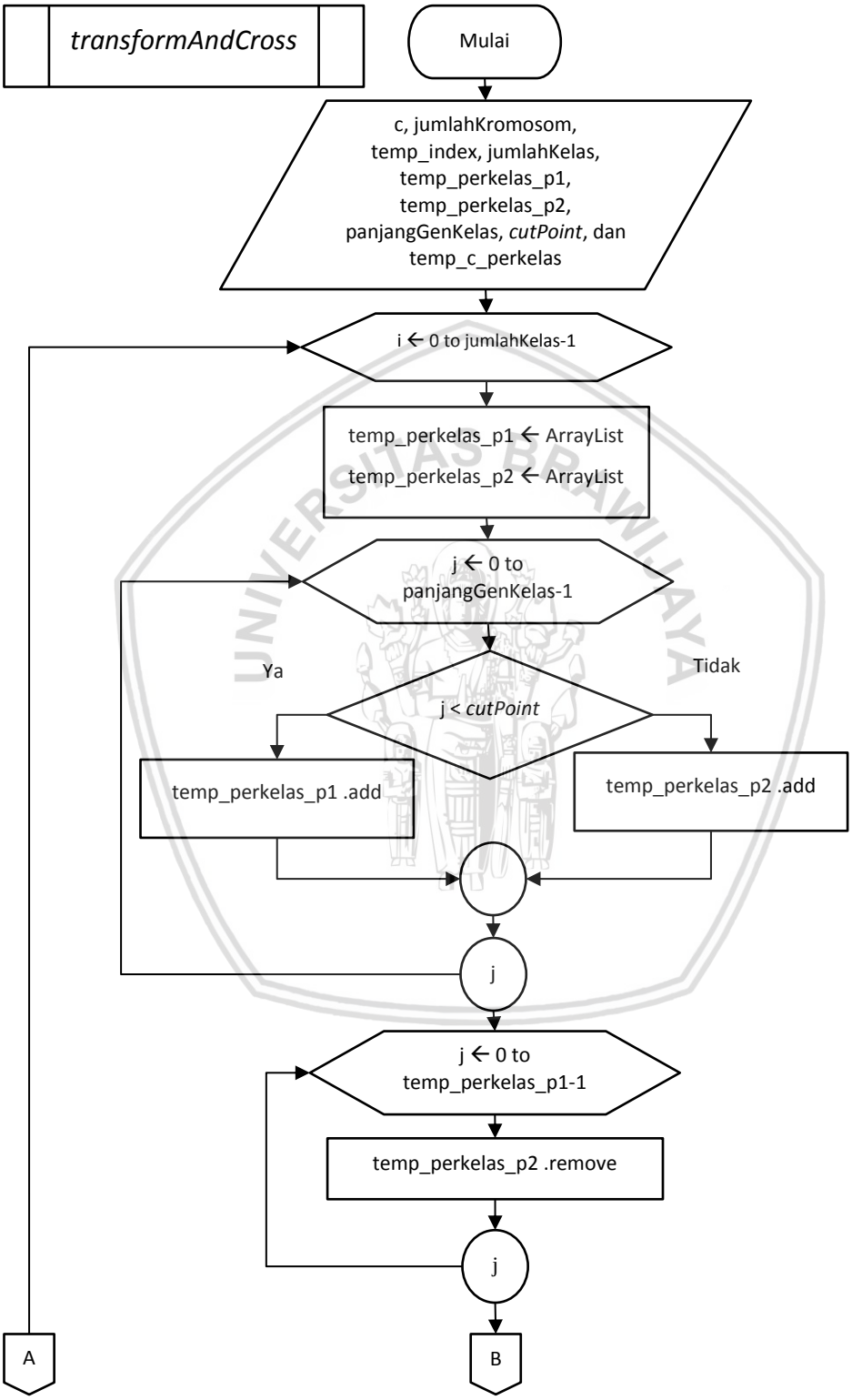


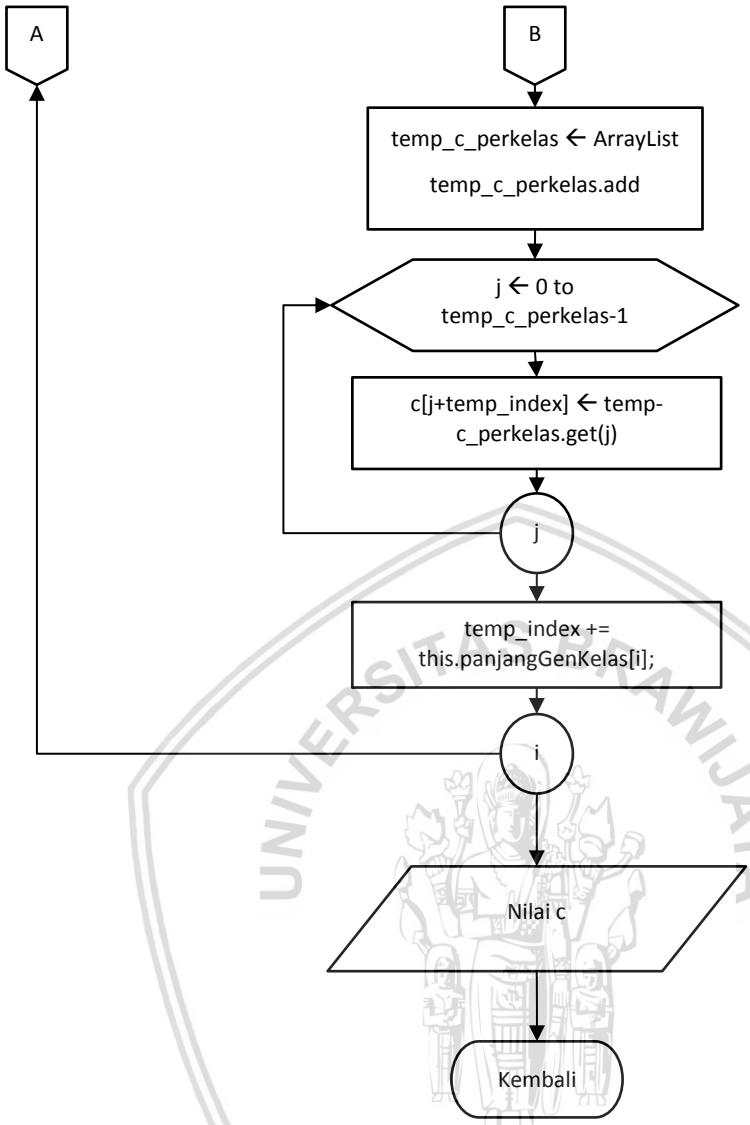
Gambar 4.5 Blok Diagram *One-Cut Point Crossover*

Proses yang dilakukan dalam *transformAndCross* pada Gambar 4.6 antara lain:

1. Sistem menerima masukan berupa c, jumlahKromosom, temp_index, jumlahKelas, temp_perkelas_p1, temp_perkelas_p2, panjangGenKelas, *cutPoint*, dan temp_c_perkelas.
2. Perulangan variabel *i* sebanyak kurang dari jumlahKelas yang berisi inisialisasi variabel temp_perkelas_p1 dan temp_perkelas_p2.
3. Perulangan variabel *j* sebanyak kurang dari panjangGenKelas yang berisi, apabila nilai variabel *j* kurang dari *cutPoint* maka nilai variabel temp_perkelas_p1 bertambah. Jika tidak maka nilai dari variabel temp_perkelas_p2 bertambah.
4. Perulangan variabel *j* untuk menghapus nilai isi dari p2 yang ada di p1.
5. Penggabungan p1 dan p2 pada variabel temp_c_perkelas.
6. Perulangan variabel *j* untuk menggabungkan p1 dan p2 menjadi satu kromosom.

7. Penambahan nilai pada variabel temp_index.
8. Pengembalian nilai pada variabel *c* dan didapatkan hasil dari *transformAndCross*.





Gambar 4.6 Blok Diagram *transformAndCross*

4.2.3.3 Mutasi

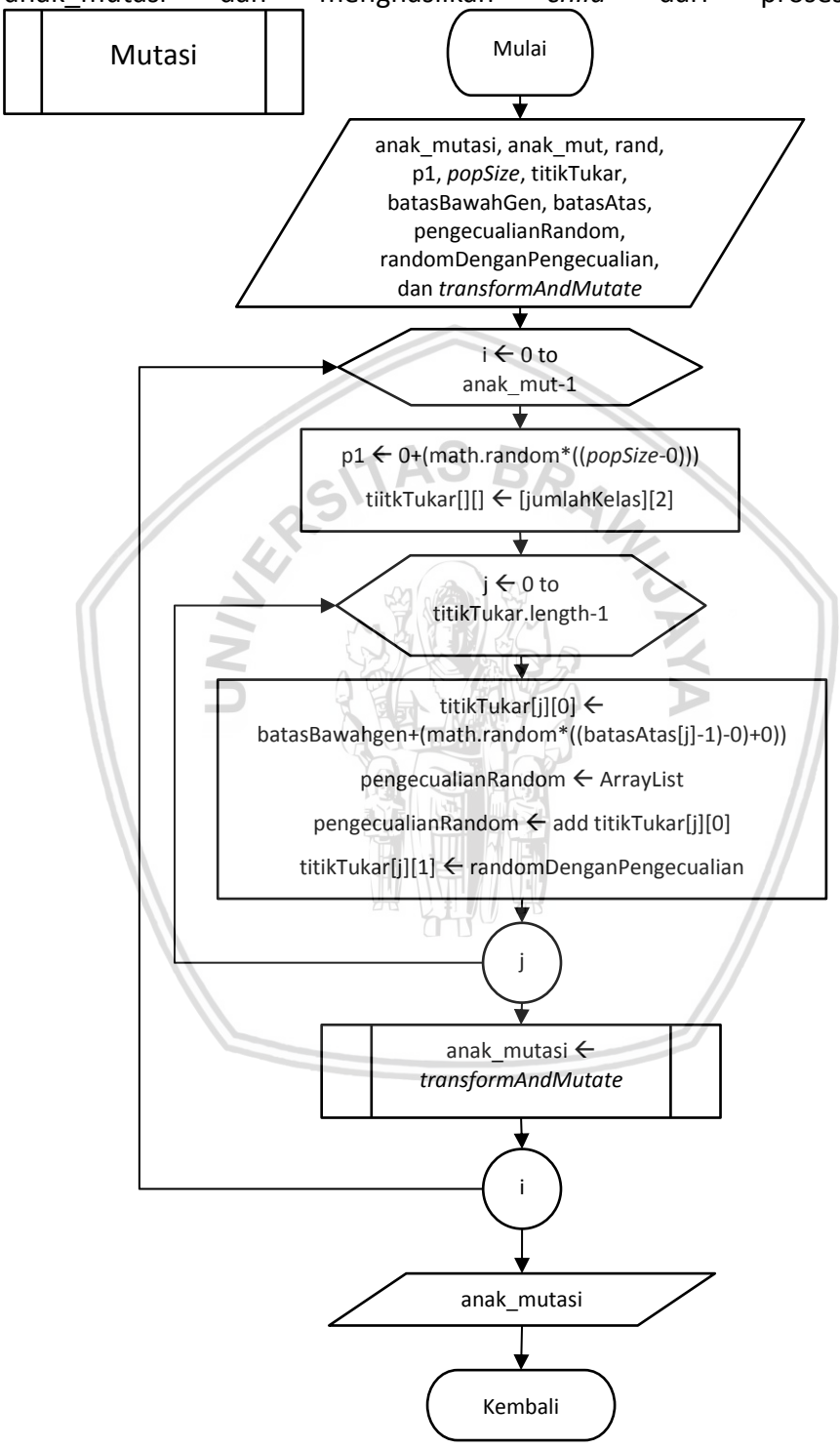
Proses mutasi merupakan salah satu proses reproduksi yang hanya menggunakan satu *parent* dalam menghasilkan *child*. Mutasi yang digunakan adalah *reciprocal exchange mutation*, yaitu menukarkan *allele* pada dua posisi gen yang berbeda. Blok diagram mutasi dapat dilihat pada Gambar 4.7.

Proses yang dilakukan dalam mutasi pada Gambar 4.7 antara lain:

1. Sistem menerima masukan berupa anak_mutasi, anak_mut, rand, p1, *popSize*, titikTukar, batasBawahGen, batasAtas, pengecualian*Random*, *random*DenganPengecualian, dan *transformAndMutate*.
2. Perulangan variabel *i* sebanyak kurang dari anak_mut yang berisi inisialisasi variabel p1, titikTukar, perulangan variabel *j*, dan inisialisasi anak_mutasi[*i*].
3. Perulangan variabel *j* sebanyak kurang dari panjang titikTukar yang berisi inisialisasi array titikTukar[*j*][1], pengecualian*Random*, penambahan array

titikTukar ke dalam pengecualian*Random*, dan inisialisasi array titikTukar[*j*][1].

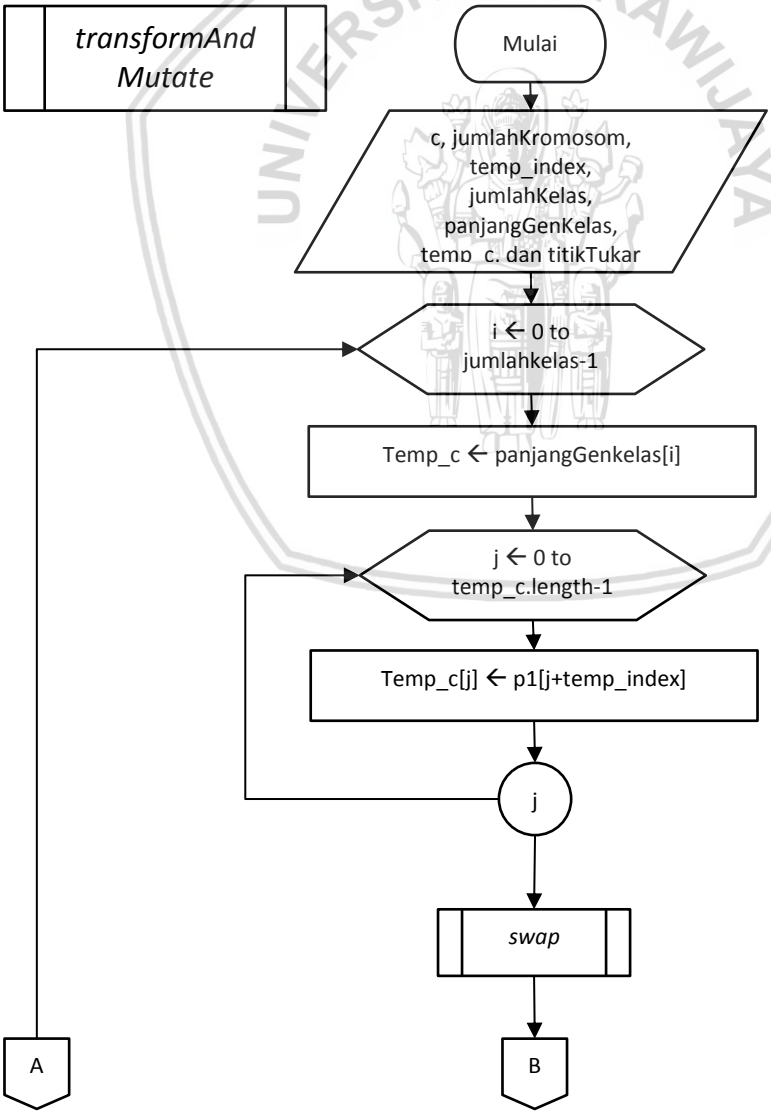
4. Setelah perulangan variabel *i* terpenuhi, maka anak dikembalikan nilai anak_mutasi dan menghasilkan *child* dari proses mutasi.

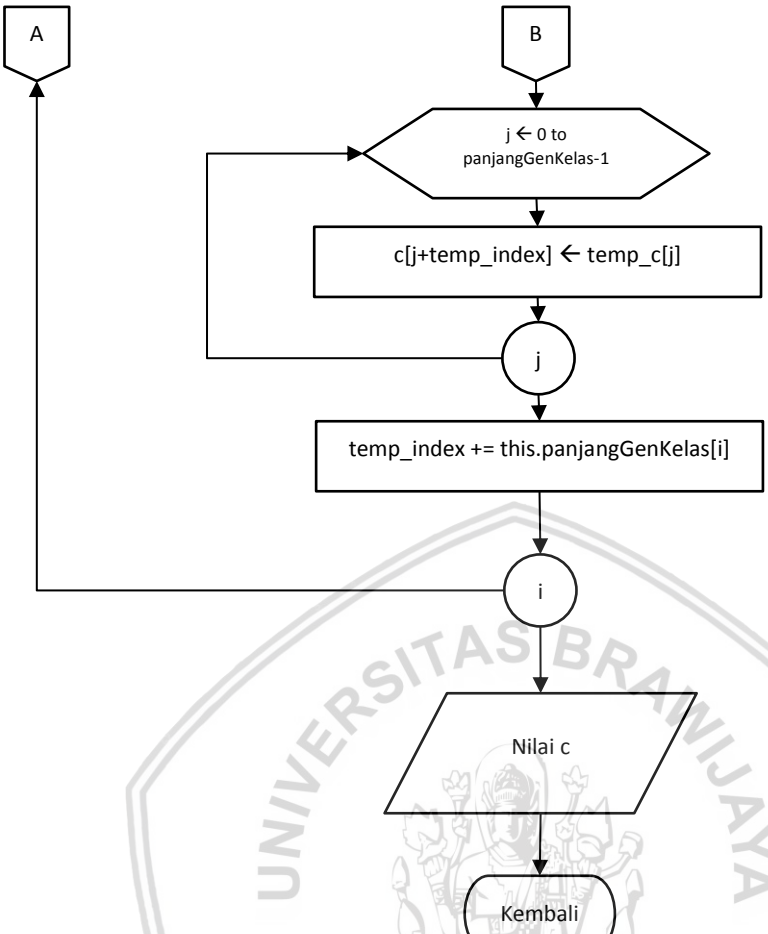


Gambar 4.7 Blok Diagram Mutasi

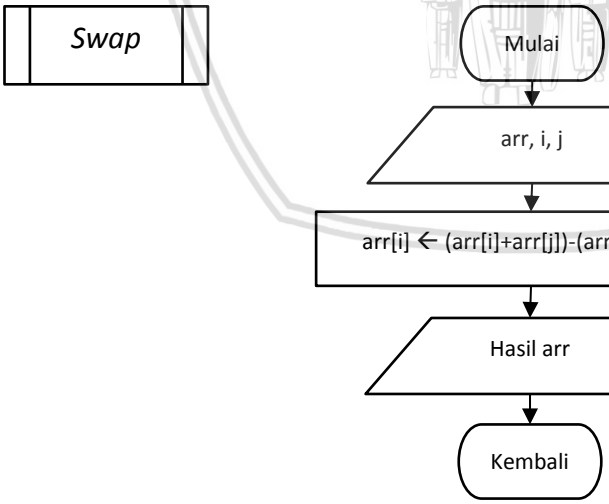
Proses yang dilakukan dalam *transformAndMutate* pada Gambar 4.8 antara lain:

1. Sistem menerima masukan berupa *c*, jumlahKromosom, temp_index, jumlahKelas, panjangGenKelas, temp_*c*, dan titikTukar.
2. Perulangan variabel *i* sebanyak kurang dari jumlahKelas yang berisi inisialisasi variabel temp_*c*.
3. Perulangan variabel *j* sebanyak kurang dari panjang temp_*c* yang berisi inisialisasi array temp_*c*[*j*].
4. Memanggil method *swap*.
5. Perulanga variabel *j* sebanyak kurang dari panjangGenKelas yang berisi inisialisasi array *c*[*j*+temp_index].
6. Penambahan nilai pada variabel temp_index.
7. Pengembalian nilai pada variabel *c* dan didapatkan hasil dari *transformAndMutate*.





Gambar 4.8 Blok Diagram transformAndMutate



Gambar 4.9 Blok Diagram *Swap*

Proses yang dilakukan dalam *swap* pada Gambar 4.9 antara lain:

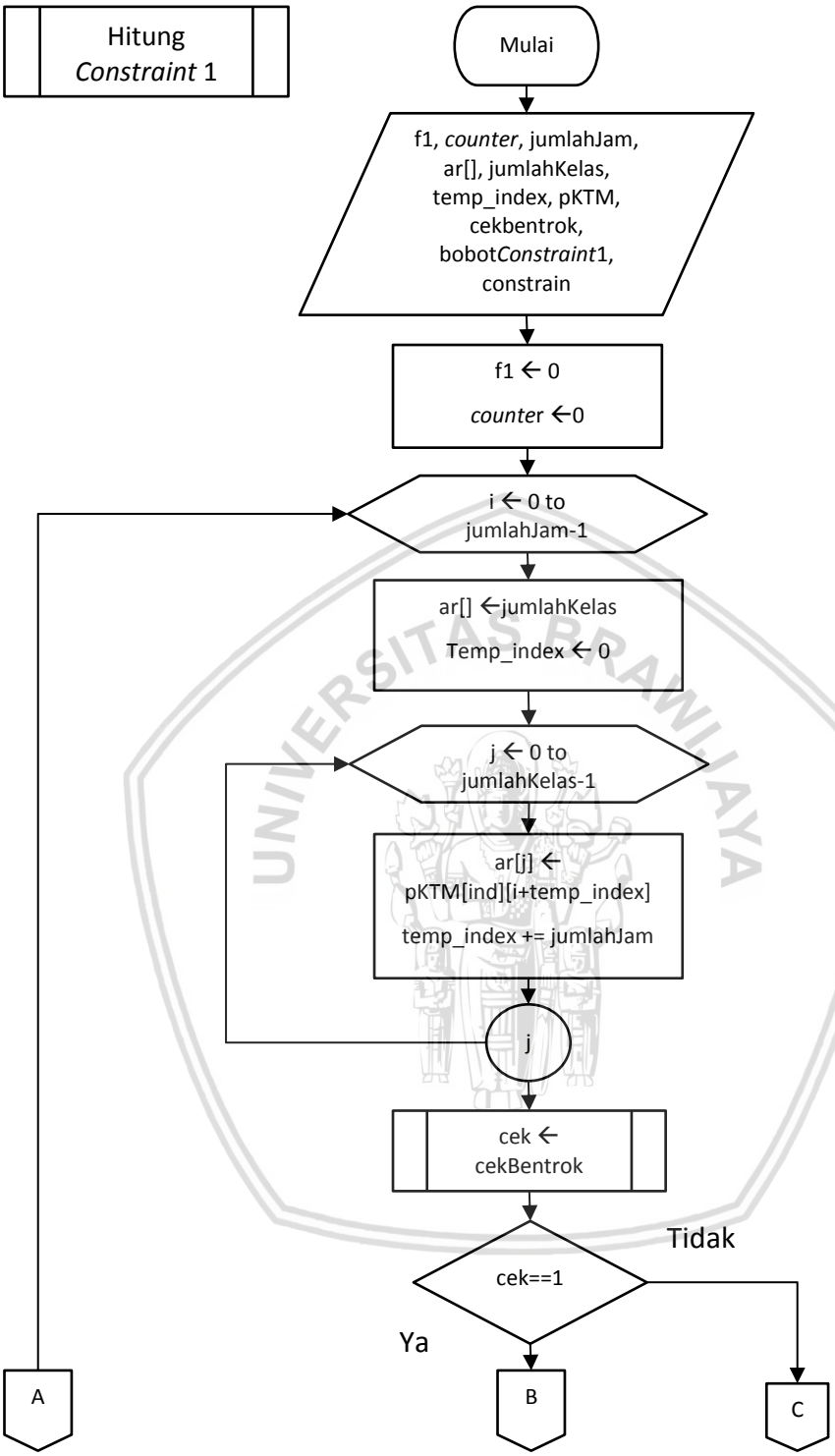
1. Sistem menerima masukan berupa variabel arr, *i*, dan *j*.
2. Inisialisasi array arr[*i*] untuk mendapatkan hasil *swap* dua buah gen dalam proses mutasi.

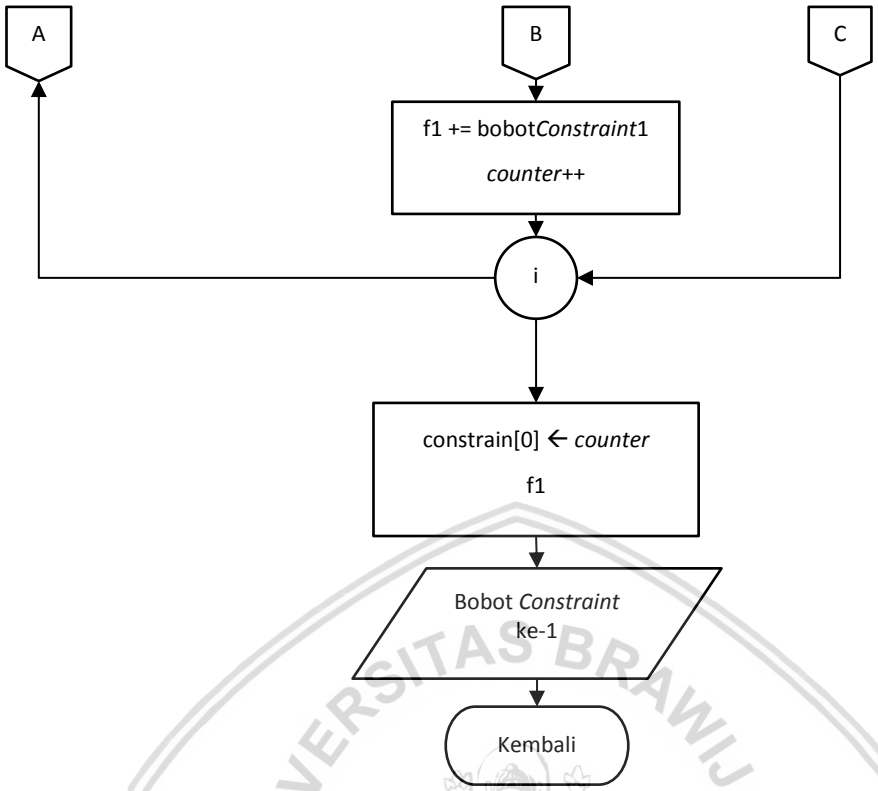
4.2.4 Hitung *Constraint* Ke-1

Nilai *fitness* yang dihasilkan diperoleh dari seberapa banyak jumlah pelanggaran terhadap *constraint* yang terjadi. Sehingga perlu adanya perhitungan *constraint* untuk mengetahui seberapa layak individu tersebut digunakan sebagai solusi. Individu dapat dikatakan melanggar *constraint* ke-1 apabila guru mengajar lebih dari satu kelas atau mengajar di lain kelas pada waktu yang sama (bentrok). Blok diagram pengecekan *constraint* ke-1 dapat dilihat pada Gambar 4.10.

Proses yang dilakukan dalam hitung c*onstraint* ke-1 pada Gambar 4.11 antara lain:

1. Sistem menerima masukan berupa f1, *counter*, jumlahJam, ar[], jumlahKelas, temp_index, pKTM, cekbentrok, bobot*Constraint*1, constrain.
2. Inisialisasi variabel f1 dan *counter*.
3. Perulangan variabel *i* sebanyak kurang dari jumlahJam yang berisi inisialisasi array ar[] dan temp_index.
4. Perulangan variabel *j* sebanyak kurang dari jumlahKelas yang berisi inisialisasiarray ar dengan indeks *j* dan penambahan nilai pada variabel temp_index.
5. Jika method cekBentrok sama dengan 1 maka bobot *constraint* pada variabel f1 bertambah, serta variabel *counter* juga bertambah. Variabel *counter* ini berfungsi untuk menghitung berapa banyak pelanggaran pada *constraint* ke-1.
6. Menginisialisasi variabel constrain indeks ke 0 dan mengembalikan nilai f1.
7. Sistem menghasilkan keluaran berupa bobot atau nilai dari *constraint* ke-1.

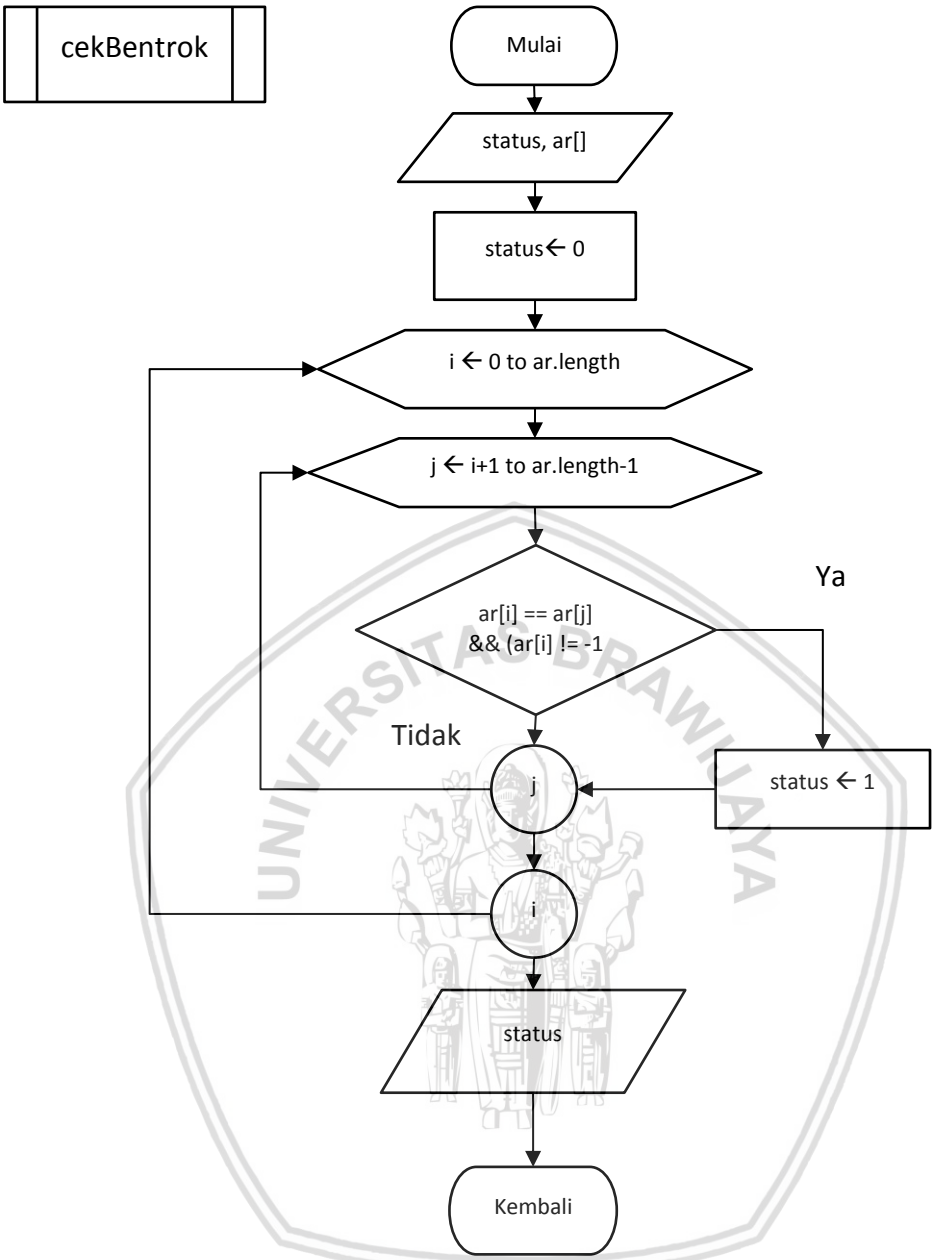

Hitung
Constraint 1
Mulai
f1, counter, jumlahJam,
ar[], jumlahKelas,
temp_index, pKTM,
cekbentrok,
bobotConstraint1,
constrain
f1 ← 0
counter ←0
i ← 0 to
jumlahJam-1
ar[] ←jumlahKelas
Temp_index ← 0
j ← 0 to
jumlahKelas-1
ar[j] ←
pKTM[ind][i+temp_index]
temp_index += jumlahJam
j
cek ←
cekBentrok
Tidak
cek==1
Ya
A
B
C



Gambar 4.10 Blok Diagram Hitung *Constraint* Ke-1

Proses yang dilakukan dalam cekBentrok pada Gambar 4.11 antara lain:

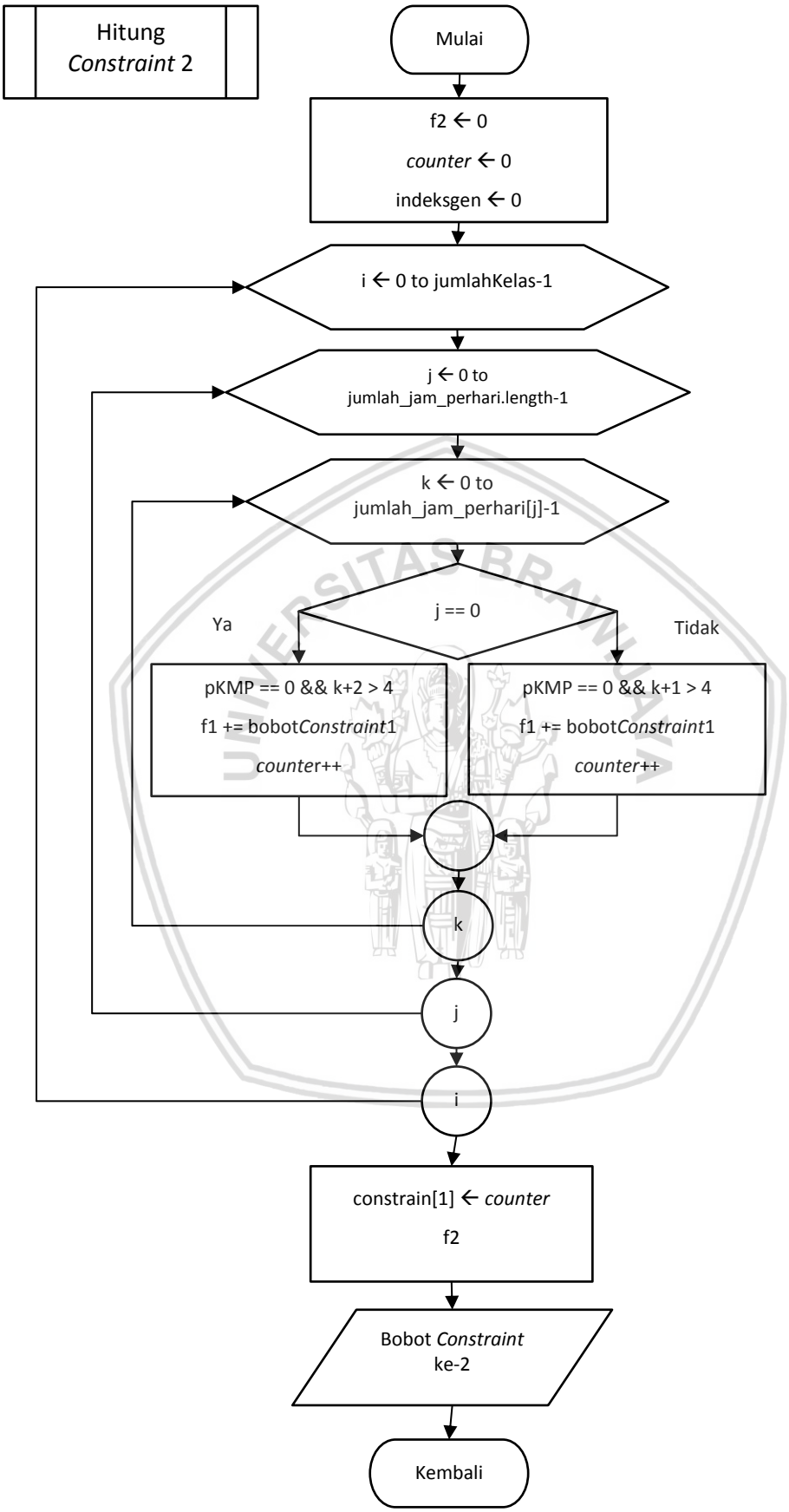
1. Sistem menerima masukan berupa variabel status dan ar.
2. Perulangan variabel *i* sebanyak kurang dari panjang variabel ar.
3. Perulangan variabel *j* sebanyak panjang variabel yang dimulai dari indeks *i*+1 yang berisi, apabila ar[*i*] == ar[*j*] && (ar[*i*] != -1 || ar[*j*] != -1 maka variabel status bernilai 1. Yang menandakan bahwa telah terjadi bentrok.
4. Pengembalian nilai status, dan sistem menghasilkan keluaran berupa jumlah bentrok.



Gambar 4.11 Blok Diagram cekBentrok

4.2.5 Hitung *Constraint* Ke-2

Individu dapat dikatakan melanggar *constraint* ke-2 apabila mata pelajaran Penjaskes OR tidak berada pada jam pelajaran ke 1-4. Blok diagram perhitungan *constraint* ke-2 dapat dilihat pada Gambar 4.12.



Gambar 4.12 Blok Diagram Hitung *Constraint* Ke-2

Proses yang dilakukan dalam hitung c*onstraint* ke-2 pada Gambar 4.12 antara lain:

1. Sistem menerima masukan berupa f2, *counter*, indeksGen, jumlahkelas, jumlah_jam_perhari, pKMP, bobot*Constraint*2, dan contrain.
2. Inisialisasi awal variabel f2, *counter*, indeksGen.
3. Perulangan variabel *i* sebanyak kurang dari jumlahKelas
4. Perulangan variabel *j* sebanyak kurang dari panjang jumlah_jam_perhari.
5. Perulangan variabel *k* sebanyak kurang dari jumlah_jam_perhari dengan indeks *j*.
6. Apabila variabel *j* sama dengan nol, kode mata pelajaran sama dengan nol, dan lebih dari jam ke-4 maka bobot *constraint* pada variabel f2 bertambah, serta variabel *counter* juga bertambah. Variabel *counter* ini berfungsi untuk menghitung berapa banyak pelanggaran pada *constraint* ke-2.
7. Jika tidak, kode mata pelajaran sama dengan nol, dan lebih dari jam ke-4 maka bobot *constraint* pada variabel f2 bertambah, serta variabel *counter* juga bertambah.
8. Menginisialisasi variabel constrain indeks ke 1 dan mengembalikan nilai f2.
9. Sistem menghasilkan keluaran berupa bobot atau nilai dari *constraint* ke-2.

4.2.6 Hitung *Constraint* ke-3

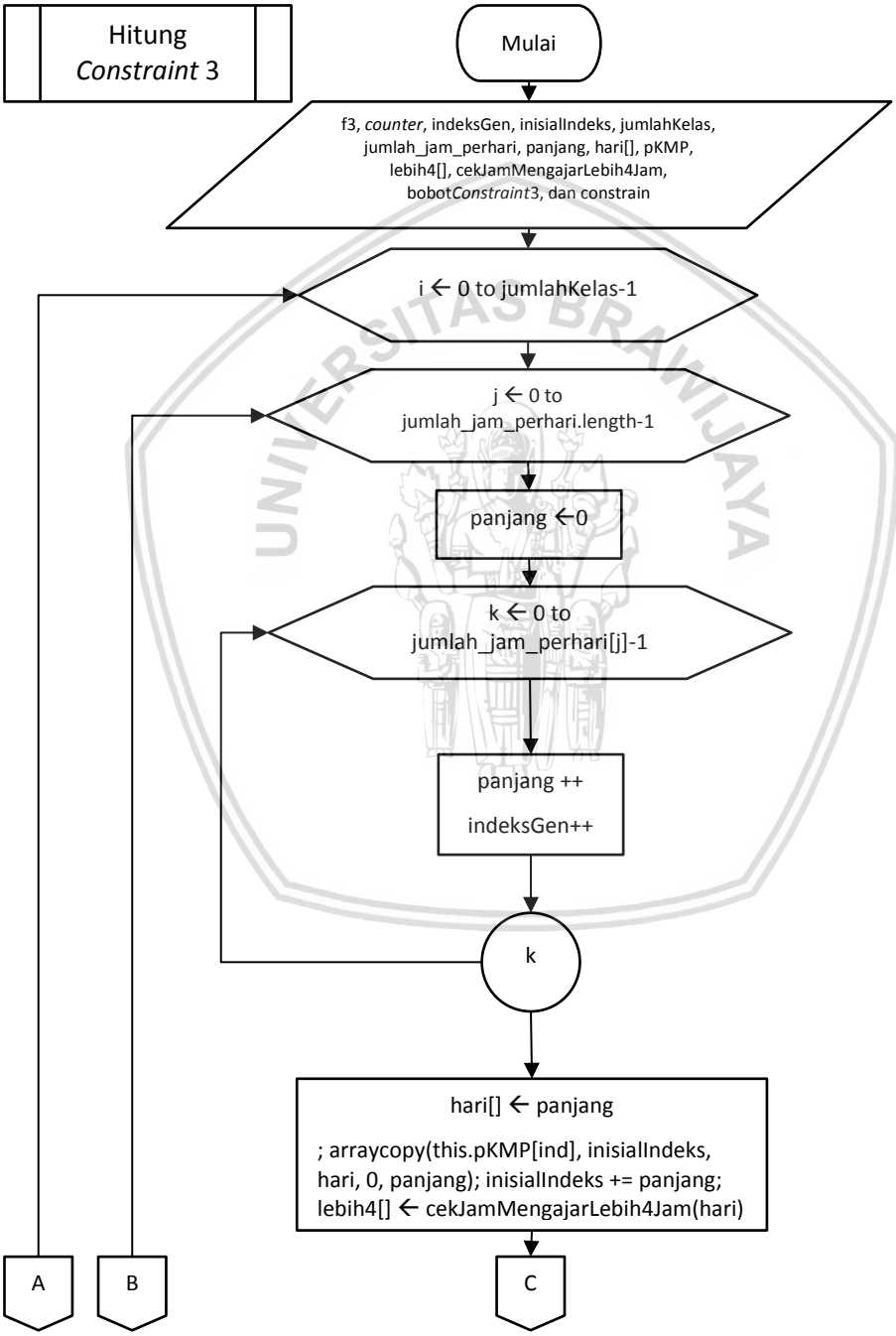
Individu dapat dikatakan melanggar *constraint* ke-3 apabila guru yang mengajar lebih dari empat jam pelajaran dalam mata pelajaran yang sama dan di hari yang sama. Blok diagram perhitungan *constraint* ke-3 dapat dilihat pada Gambar 4.13.

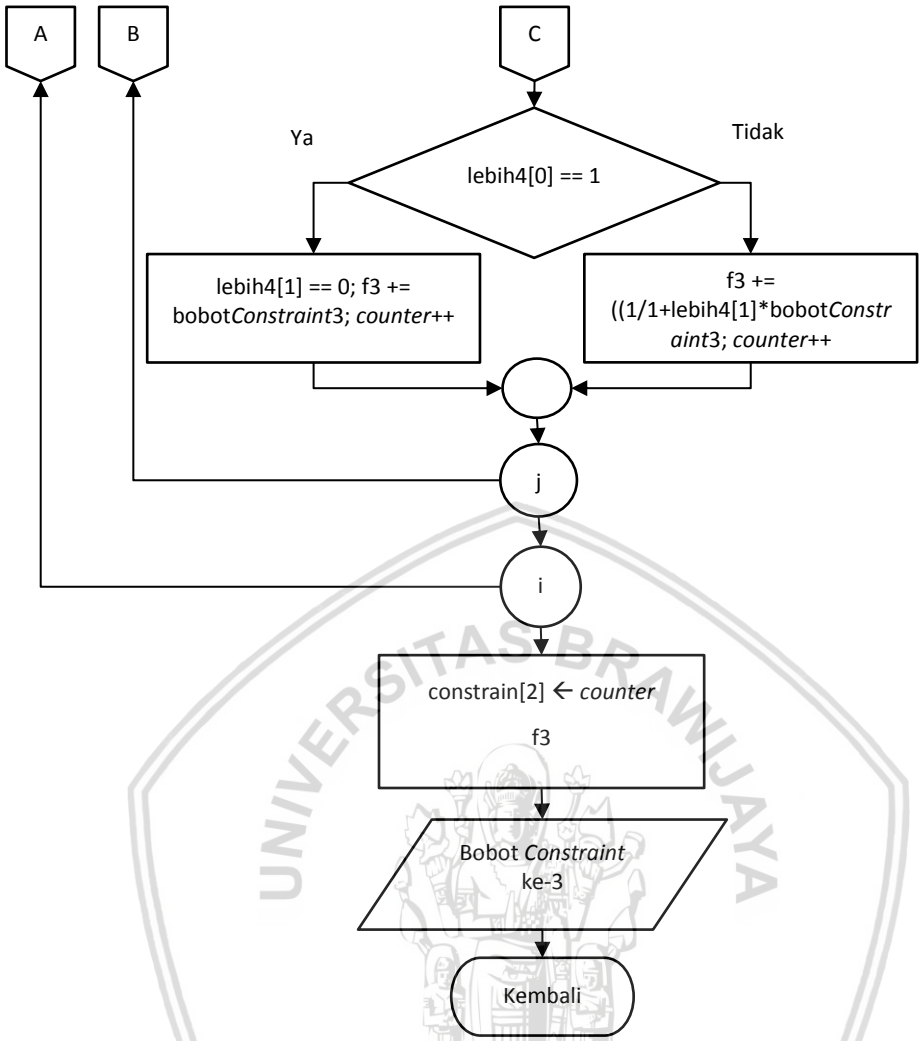
Proses yang dilakukan dalam hitung c*onstraint* ke-3 pada Gambar 4.13 antara lain:

1. Sistem menerima masukan berupa f3, *counter*, indeksGen, inisialIndeks, jumlahKelas, jumlah_jam_perhari, panjang, hari[], pKMP, lebih4[], cekJamMengajarLebih4Jam, bobot*Constraint*3, dan constrain.
2. Inisialisasi awal variabel f3, *counter*, indeksGen, dan inisialIndeks.
3. Perulangan variabel *i* sebanyak kurang dari jumlahKelas.
4. Perulangan variabel *j* sebanyak kurang dari panjang jumlah_jam_perhari yang berisi inisialisasi variabel panjang.
5. Perulangan variabel *k* sebanyak kurang dari jumlah_jam_perhari yang berisi *increment* variabel panjang dan indeksGen.
6. Inisialisasi hari[], inisialIndeks, lebih4[], dan meng-*copy* array.
7. Apabila variabel lebih4 indeks 0 sama dengan satu dan indeks 1 sama dengan nol maka bobot *constraint* pada variabel f3 bertambah, serta variabel *counter*

juga bertambah. Variabel *counter* ini berfungsi untuk menghitung berapa banyak pelanggaran pada *constraint* ke-3.

8. Jika tidak, kode mata pelajaran sama dengan nol, dan lebih dari jam ke-4 maka bobot *constraint* pada variabel f3 bertambah, dengan perhitungan yang berbeda. Serta variabel *counter* juga bertambah.
9. Menginisialisasi variabel constrain indeks ke 2 dan mengembalikan nilai f3.
10. Sistem menghasilkan keluaran berupa bobot atau nilai dari *constraint* ke-3.



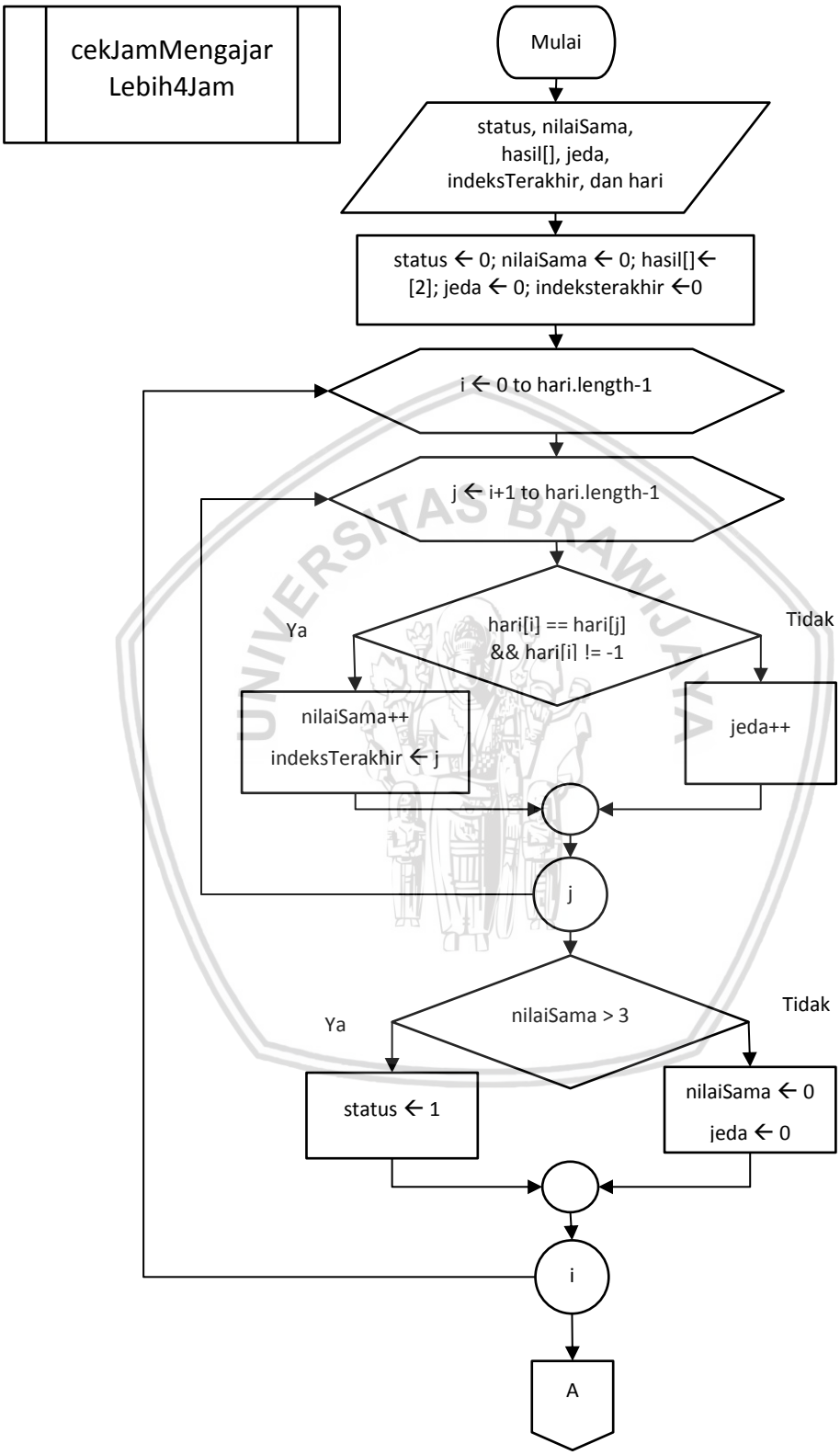


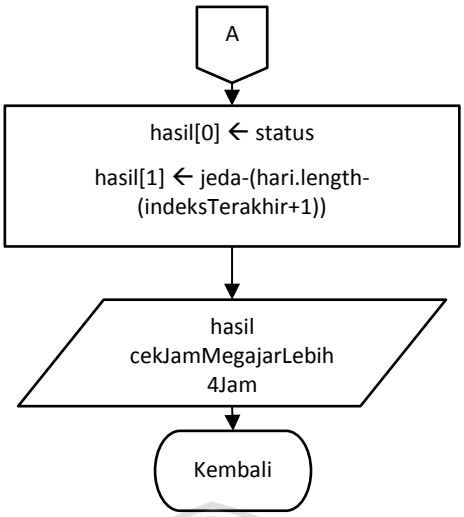
Gambar 4.13 Blok Diagram Hitung *Constraint* Ke-3

Proses yang dilakukan dalam cekJamMengajarLebih4Jam pada Gambar 4.14 antara lain:

1. Sistem menerima masukan berupa status, nilaiSama, hasil[], jeda, indeksTerakhir, dan hari.
2. Inisialisasi awal variabel status, nilaiSama, hasil[], jeda, indeksTerakhir.
3. Perulangan variabel *i* sebanyak kurang dari panjang variabel hari.
4. Perulangan variabel *j* dari indeks ke *i*+1 sampai kurang dari panjang variabel hari yang berisi, jika hari[*i*] == hari[*j*] && hari[*i*] != -1 maka *increment* pada variabel nilaiSama dan inisialisasi variabel indeksTerakhir dengan nilai *j*. Jika tidak, maka *increment* variabel jeda.
5. Jika nilaiSama > 3 maka nilai status berubah menjadi 1 yang berarti terjadi pelanggaran, jika tidak maka bernilai 0 beserta variabel jeda.
6. Nilai variabel status akan dimasukkan pada array hasil indeks ke nol, dan pelanggaran yang terdapat jeda akan dimasukkan ke indeks ke satu.

7. Pengembalian nilai variabel hasil dan sistem menghasilkan keluaran berupa hasil pelanggaran.





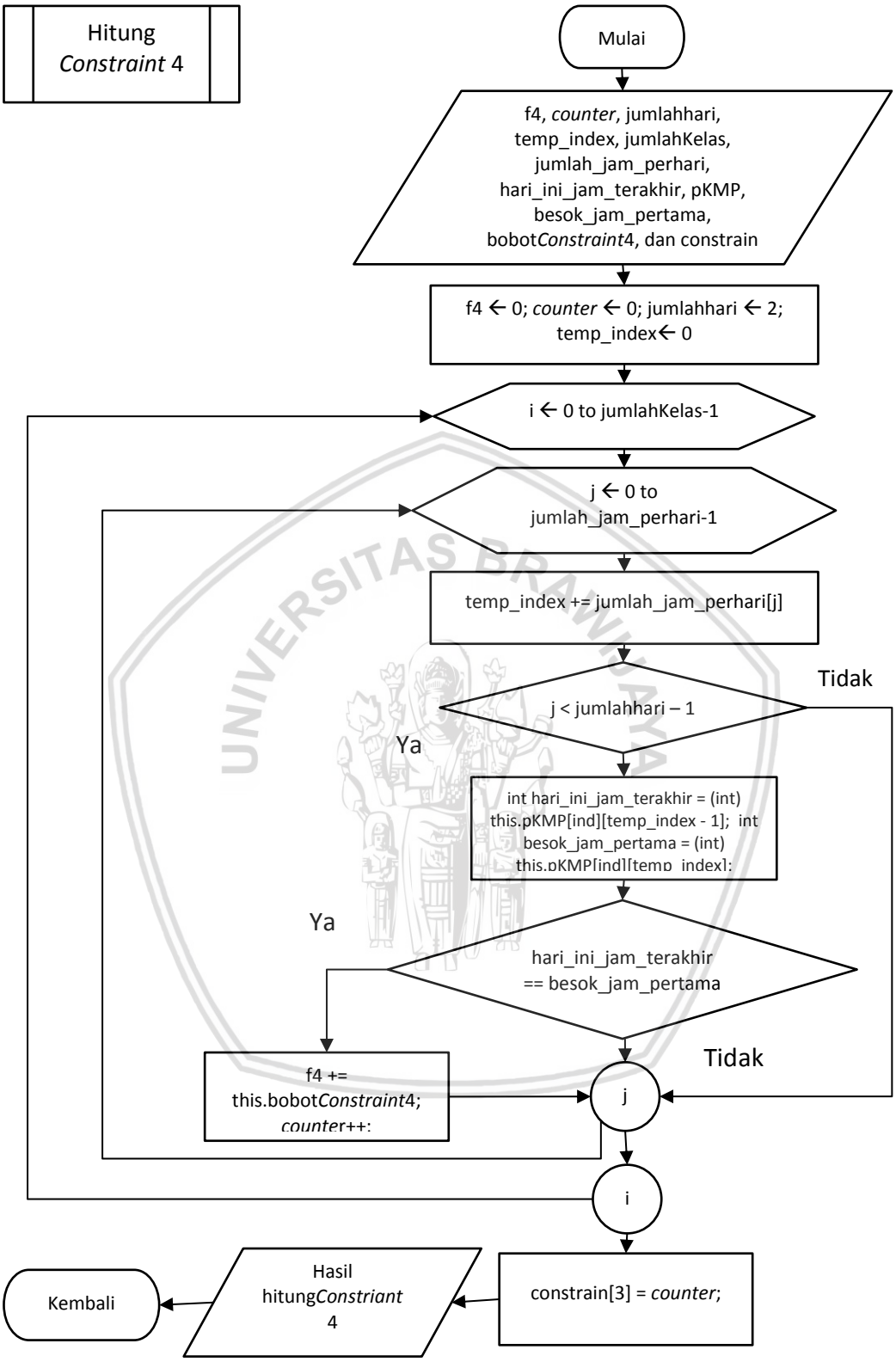
Gambar 4.14 Blok Diagram cekMengajarLebih4Jam

4.2.7 Hitung *Constraint* ke-4

Individu dapat dikatakan melanggar *constraint* ke-4 apabila dalam satu mata pelajaran yang sama pada jam kedua dan selanjutnya (sejumlah ketentuan) berada pada hari yang berbeda. Blok diagram perhitungan *constraint* ke-4 dapat dilihat pada Gambar 4.15.

Proses yang dilakukan dalam hitung *constraint* ke-4 pada Gambar 4.15 antara lain:

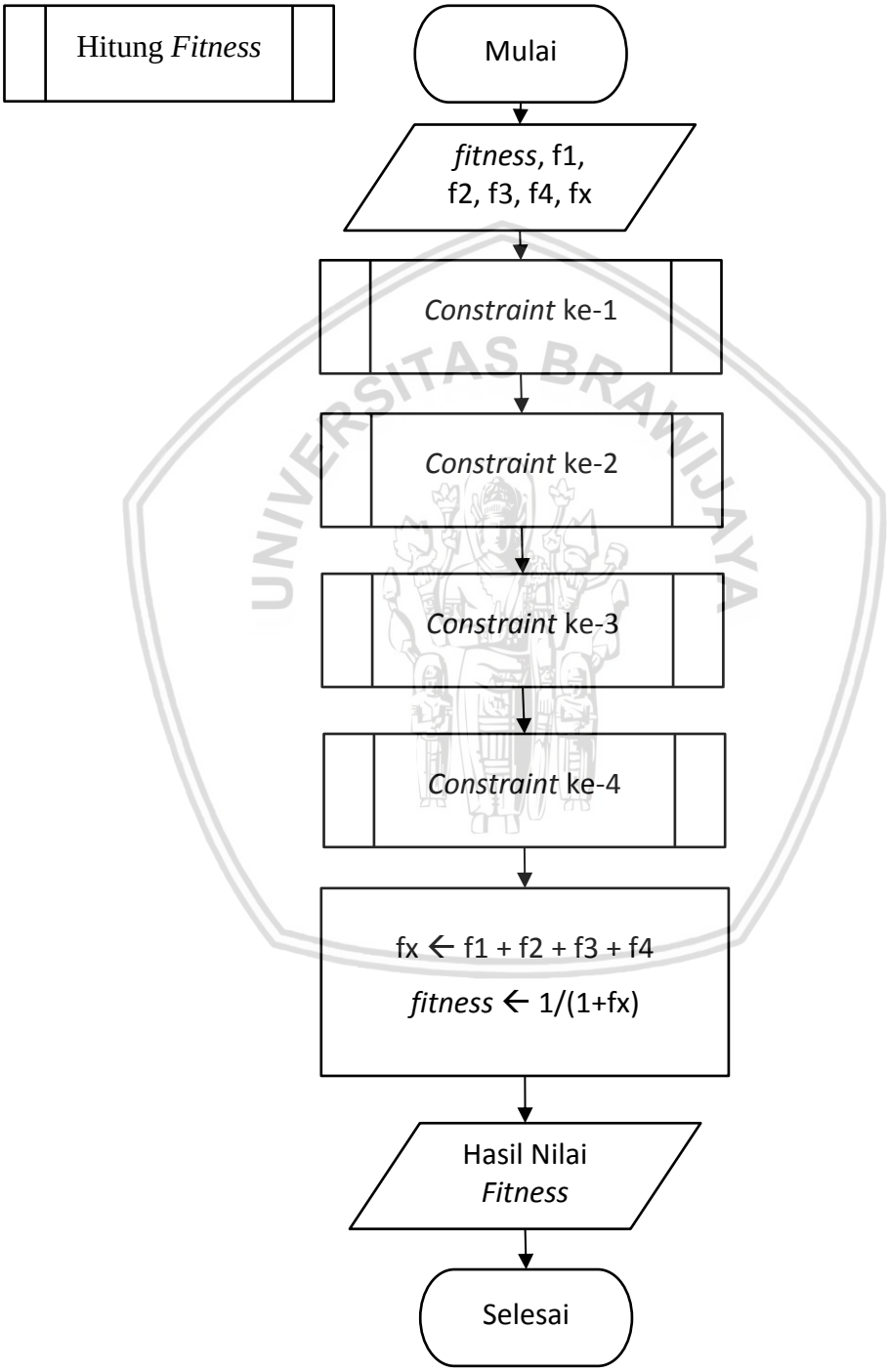
1. Sistem menerima masukan berupa f4, *counter*, jumlahhari, temp_index, jumlahKelas, jumlah_jam_perhari, hari_ini_jam_terakhir, pKMP, besok_jam_pertama, bobot*Constraint*4, dan constrain.
2. Inisialisasi awal variabel f4, *counter*, jumlahhari, temp_index.
3. Perulangan variabel *i* sebanyak kurang dari jumlahKelas.
4. Perulangan variabel *j* sebanyak kurang dari panjang variabel jumlah_jam_perhari yang berisi penambahan nilai pada variabel temp_index.
5. Jika *j* < jumlahhari-1 maka mengganti nilai variabel hari_ini_jam_terakhir dan besok_jam_pertama. Dan jika nilai variabel hari_ini_jam_terakhir dan besok_jam_pertama sama maka bobot *constraint* pada variabel f4 bertambah, serta variabel *counter* juga bertambah. Variabel *counter* ini berfungsi untuk menghitung berapa banyak pelanggaran pada *constraint* ke-4.
6. Menginisialisasi variabel constrain indeks ke 3 dan mengembalikan nilai f4.
7. Sistem menghasilkan keluaran berupa bobot atau nilai dari *constraint* ke-4.



Gambar 4.15 Blok Diagram Hitung *Constraint* Ke-4

4.2.8 Perhitungan Nilai *Fitness*

Kualitas individu dapat dlihat dari hasil nilai *fitness*nya. Nilai *fitness* didapat dari perhitungan seberapa banyak pelanggaran yang terjadi pada masing-masing *constraint*. Semakin banyak pelangaran yang terjadi pada masing-masing *constraint*, maka semakin rendah nilai *fitness* yang dihasilkan dan sebaliknya. Blok diagram perhitungan nilai *fitness* dapat dilihat pada Gambar 4.16.



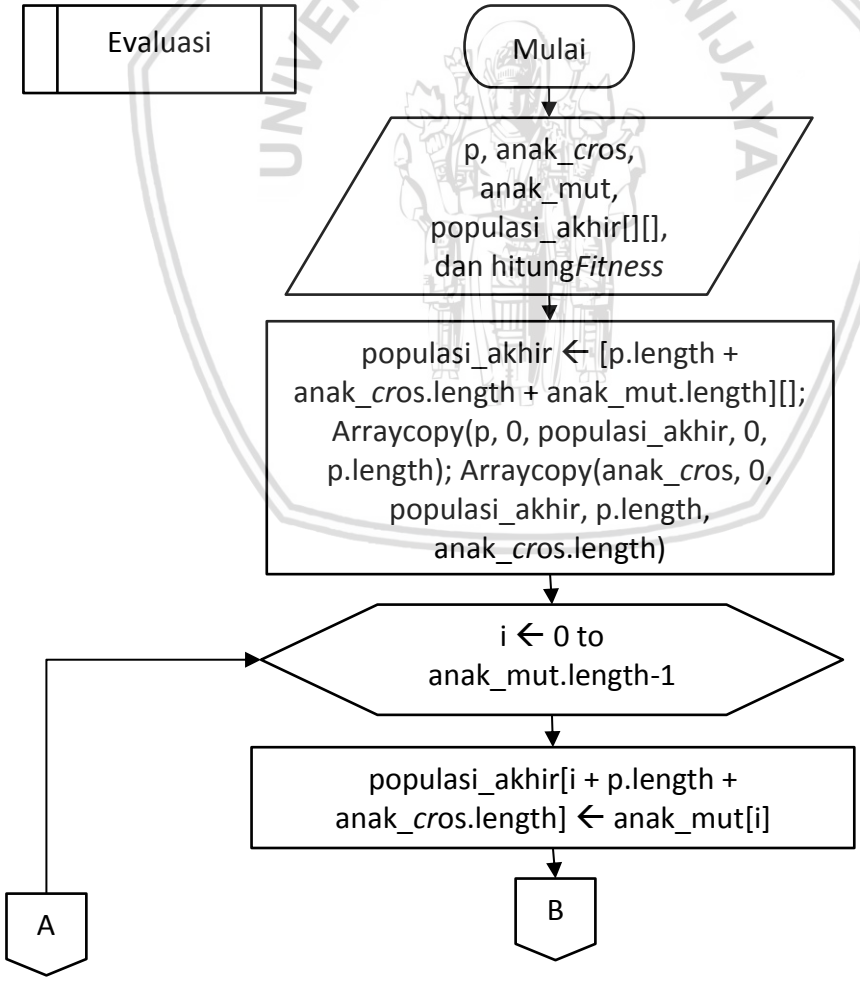
Gambar 4.16 Blok Diagram Hitung *Fitness*

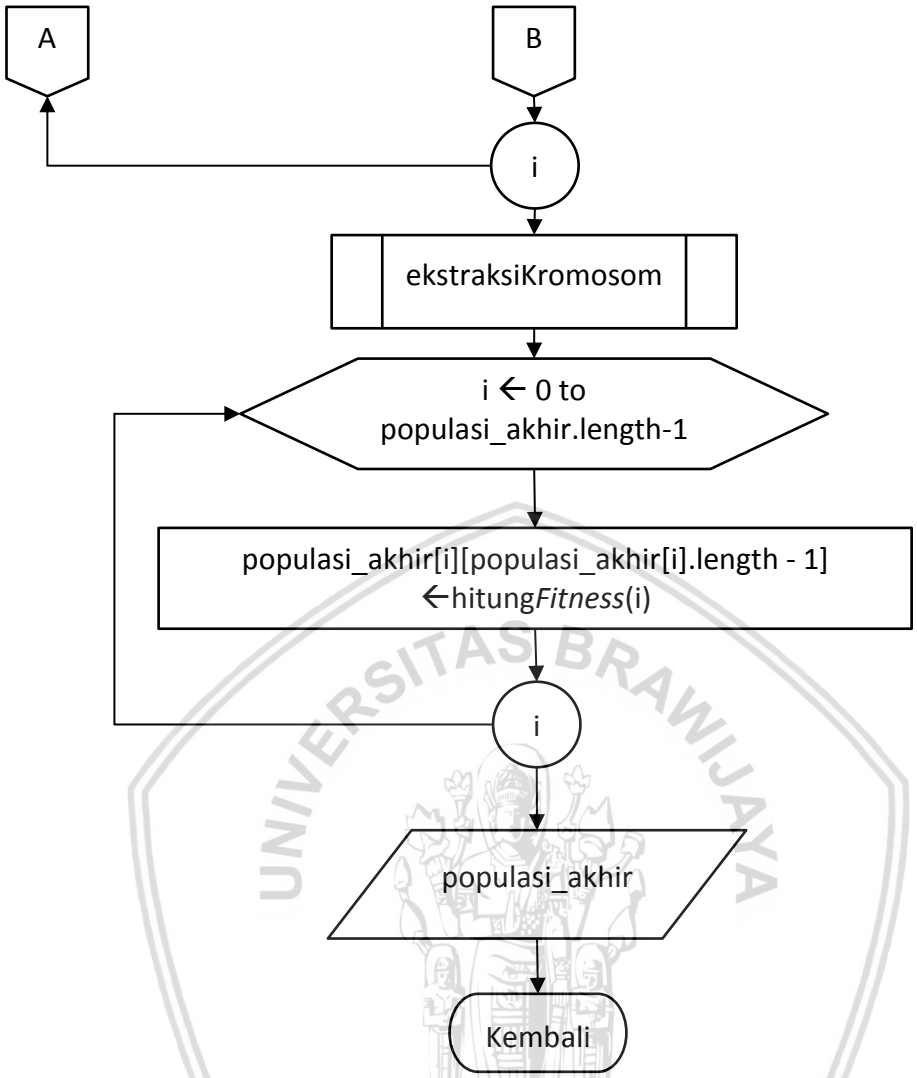
Proses yang dilakukan dalam Perhitungan Nilai *Fitness* pada Gambar 4.10 antara lain:

1. Sistem menerima masukan berupa f*itness*, f1, f2, f3, f4, fx.
2. Proses perhitungan jumlah pelanggaran *constraint* ke-1.
3. Proses perhitungan jumlah pelanggaran *constraint* ke-2.
4. Proses perhitungan jumlah pelanggaran *constraint* ke-3.
5. Proses perhitungan jumlah pelanggaran *constraint* ke-4.
6. Proses perhitungan nilai *fitness* berdasarkan jumlah pelanggaran *constraint* yang terjadi.
7. Sistem mengeluarkan hasil berupa nilai *fitness*.

4.2.9 Evaluasi

Proses evaluasi dilakukan setelah proses reproduksi. Dalam proses ini dilakukan perhitungan nilai *fitness* setiap individu dalam setiap generasinya (iterasi). Blok diagram evaluasi dapat dilihat pada Gambar 4.17.

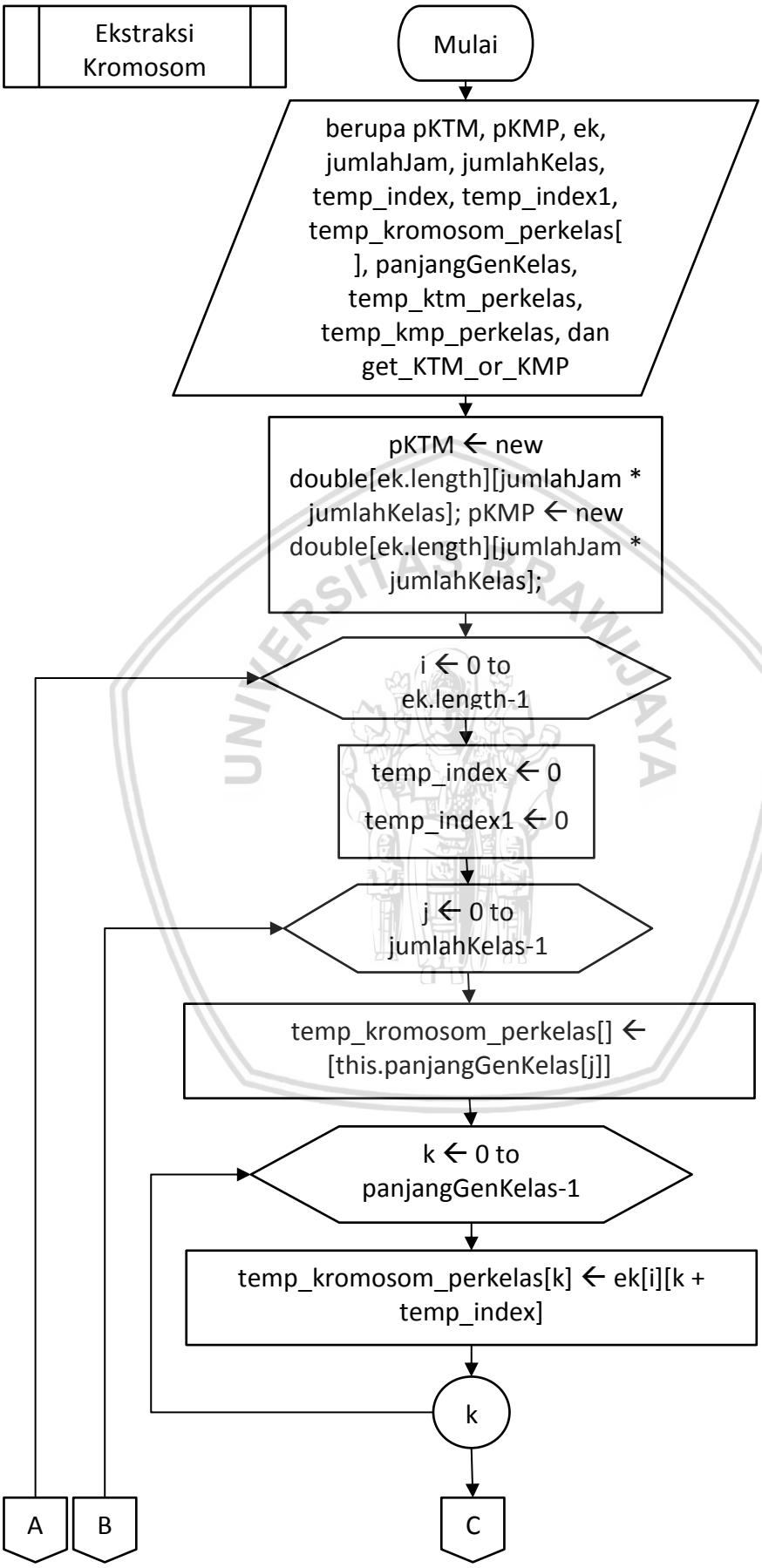


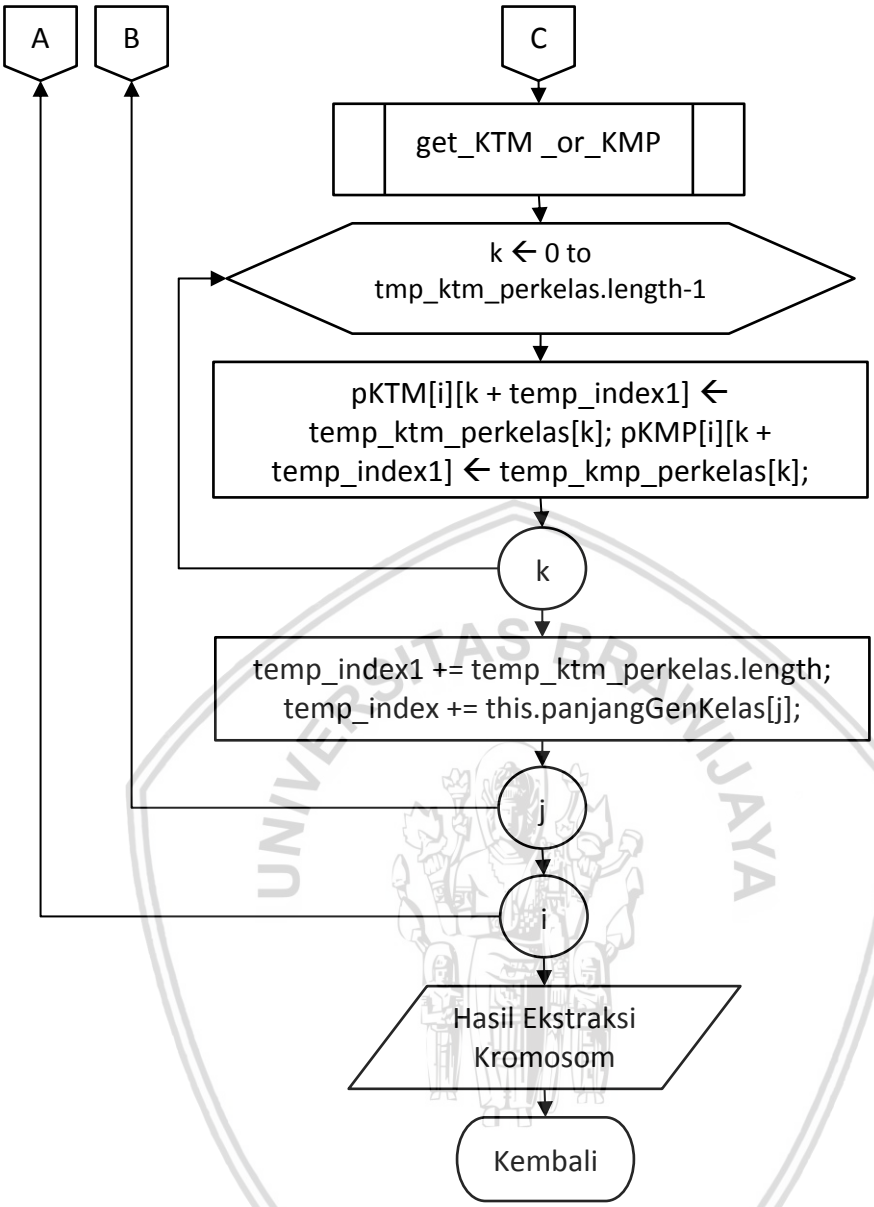


Gambar 4.17 Blok Diagram Evaluasi

Proses yang dilakukan dalam Evaluasi pada Gambar 4.17 antara lain:

1. Sistem menerima masukan berupa p, anak_*cr*os, anak_mut, populasi_akhir[][], dan hitung*Fitness*.
2. Inisialisasi varibel populasi_akhir dan meng-*copy* array.
3. Perulangan variabel *i* sebanyak kurang dari panjang variabel anak_mut yang berisi nilai populasi_akhir dengan indeks *i*+p.length+anak_*cr*os.length.
4. Memanggil ekstraksiKromosom.
5. Perulangan variabel *i* sebanyak kurang dari panjang variabel populasi_akhir yang berisi perhitungan nilai *fitness* pada populasi akhir.
6. Pengembalian nilai populasi akhir dan sistem mengeluarkan hasil evaluasi.


Ekstraksi Kromosom
Mulai
berupa pKTM, pKMP, ek, jumlahJam, jumlahKelas, temp_index, temp_index1, temp_kromosom_perkelas[ ], panjangGenKelas, temp_ktm_perkelas, temp_kmp_perkelas, dan get_KTM_or_KMP
pKTM ← new double[ek.length][jumlahJam * jumlahKelas]; pKMP ← new double[ek.length][jumlahJam * jumlahKelas];
i ← 0 to ek.length-1
temp_index ← 0
temp_index1 ← 0
j ← 0 to jumlahKelas-1
temp_kromosom_perkelas[] ← [this.panjangGenKelas[j]]
k ← 0 to panjangGenKelas-1
temp_kromosom_perkelas[k] ← ek[i][k + temp_index]
k
A
B
C

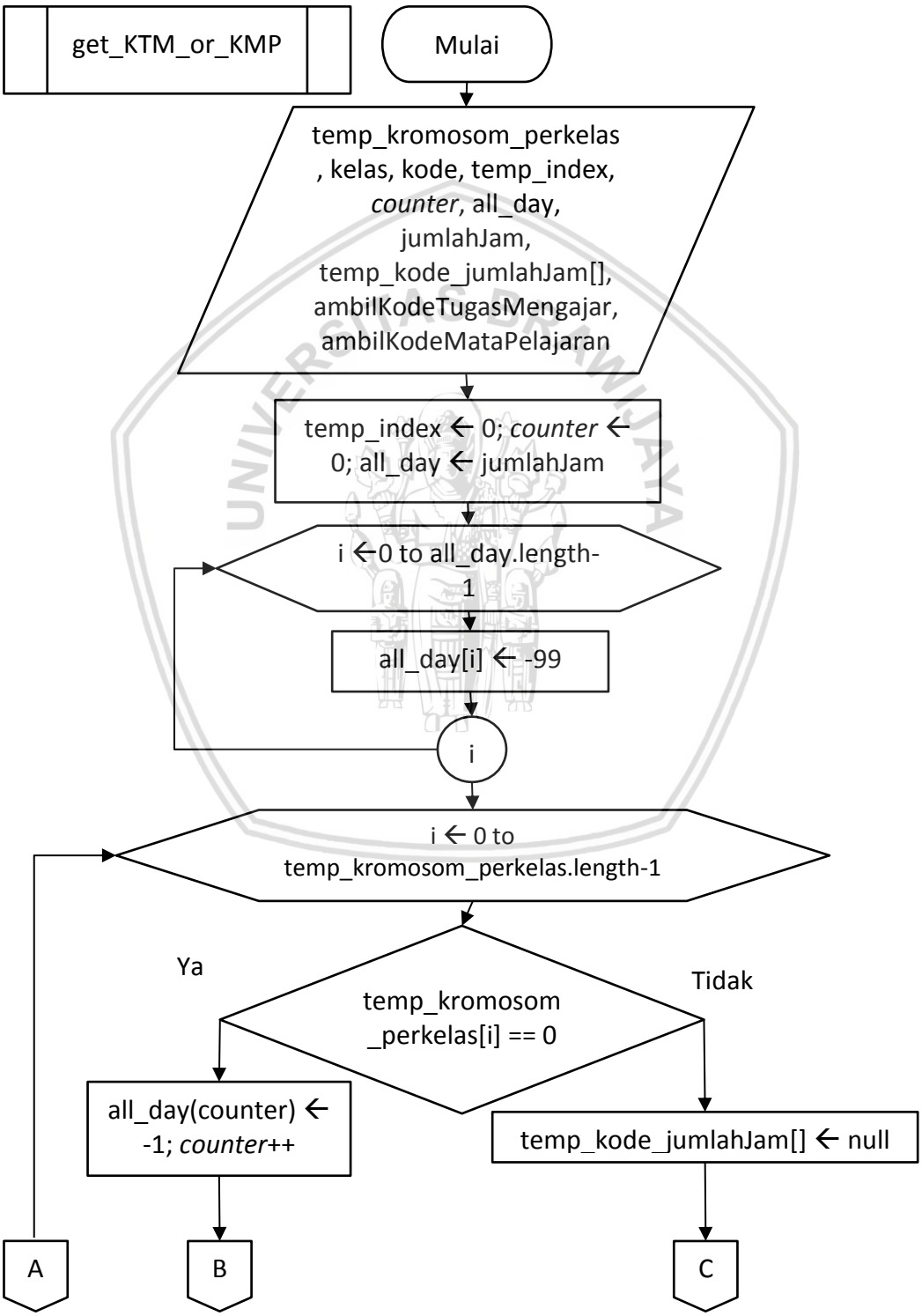


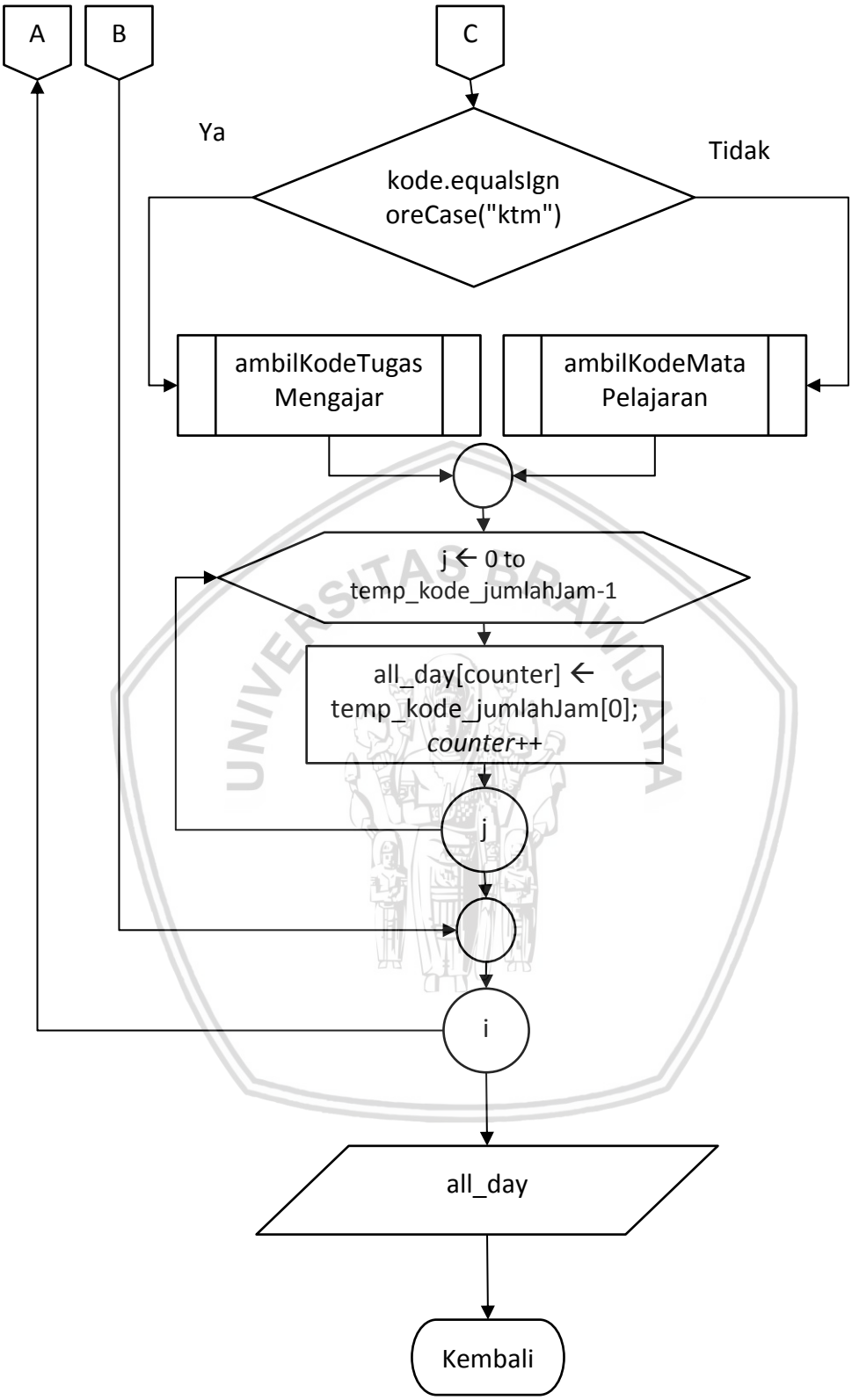
Gambar 4.18 Blok Diagram Ekstraksi Kromosom

Proses yang dilakukan dalam ekstraksi kromosom pada Gambar 4.18 antara lain:

1. Sistem menerima masukan berupa pKTM, pKMP, ek, jumlahJam, jumlahKelas, temp_index, temp_index1, temp_kromosom_perkelas[], panjangGenKelas, temp_ktm_perkelas, temp_kmp_perkelas, dan get_KTM_or_KMP.
2. Inisialisasi variabel pKTM dan pKMP.
3. Perulangan variabel *i* sebanyak kurang dari panjang variabel ek yang berisi inisialisasi variabel temp_index dan temp_index1.
4. Perulangan variabel *j* sebanyak kurang dri jumlahkelas berisi inisialisasi variabel temp_kromosom_perkelas[].

5. Perulangan variabel *k* sebanyak kurang dari panjangGenKelas[j] berisi inisialisasi variabel temp_kromosom_perkelas[*j*].
6. Memanggil method get_KTM_or_KMP.
7. Perulangan variabel *k* sebanyak kurang dari panjang variabel temp_ktm_perkelas yang berisi inisialisaasi array variabel pKTM dan pKMP.
8. Sistem mengeluarkan hasil dari ekstraksi kromosom.

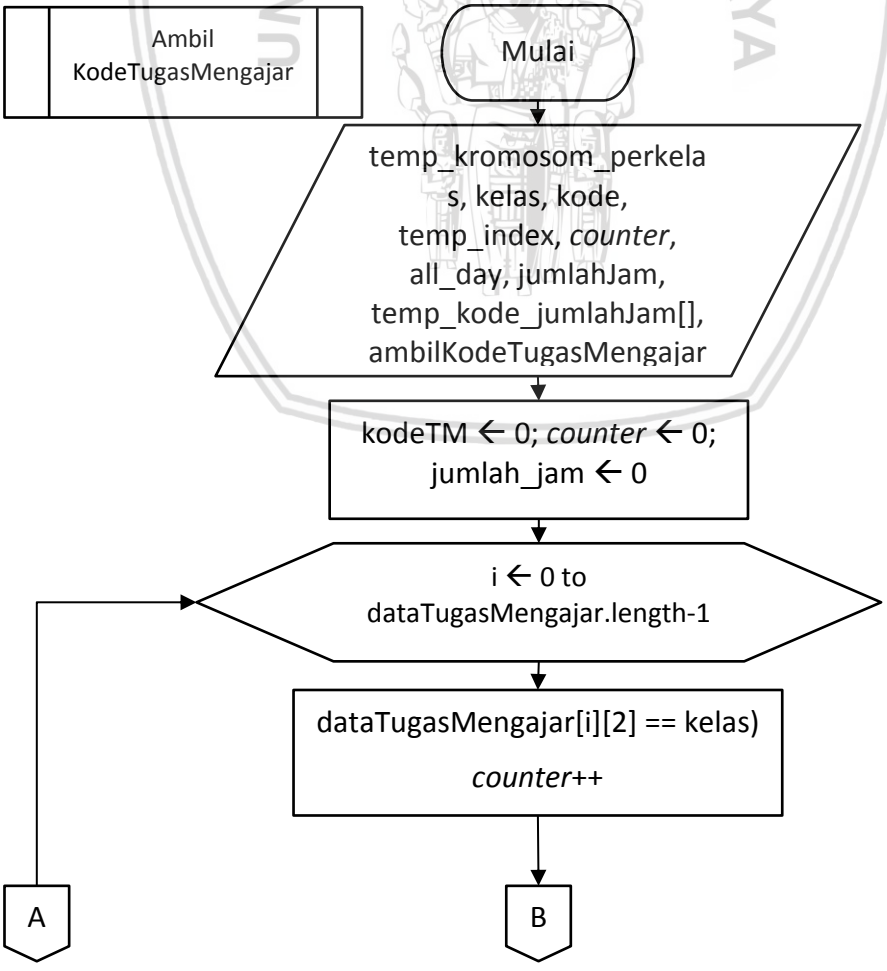


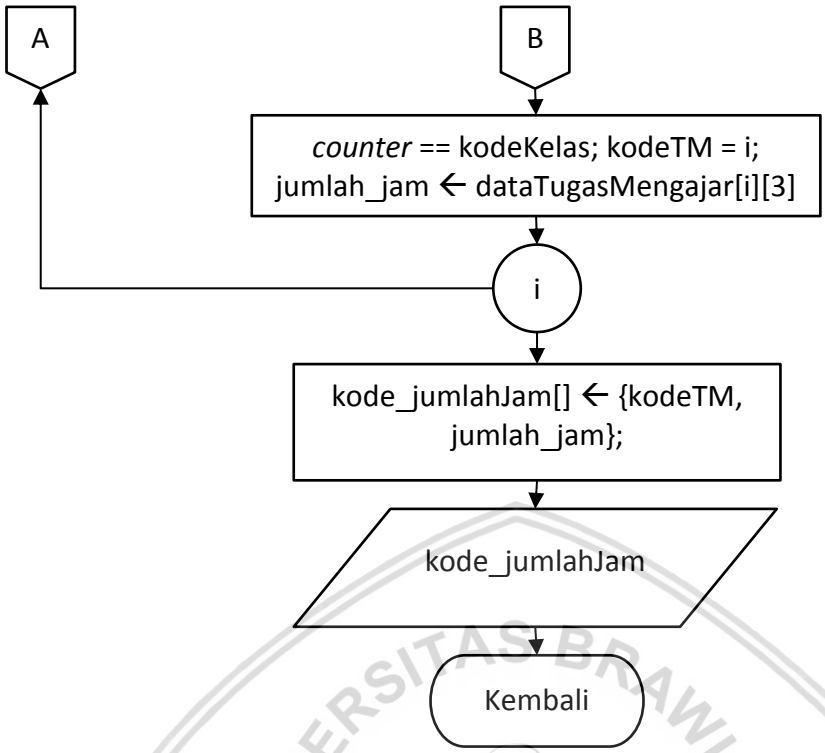


Gambar 4.19 Blok Diagram *get_KTM_or_KMP*

Proses yang dilakukan dalam *get_KTM_or_KMP* pada Gambar 4.19 antara lain:

1. Sistem menerima masukan berupa temp_kromosom_perkelas, kelas, kode, temp_index, *counter*, all_day, jumlahJam, temp_kode_jumlahJam[], ambilKodeTugasMengajar, ambilKodeMataPelajaran.
2. Inisialisai awal variabel temp_index, *counter*, dan all_day.
3. Perulangan variabel *i* sebanyak kurang dari panjang variabel all_day yang berisi inisialisasi variabel all_day indeks ke *i* bernilai -99.
4. Perulangan variabel *i* sebanyak kurang dari panjang variabel temp_kromosom_perkelas.
5. Apabila temp_kromosom_perkelas[*i*] sama dengan nol maka nilai variabel all_day dengan indeks *counter* bernilai -1. Jika tidak maka akan memanggil method ambilKodeTugasMengajar dan ambilKodeMataPelajaran.
6. Perulangan variabel *j* sebanyak kurang dari variabel temp_kode_jumlahJam dengan indeks 1 yang berisi inisialisasi variabel all_day dengan indeks *counter* dan *increment* variabel *counter*.
7. Pengembalian nilai variabel all_day dan sistem mengeluarkan hasil get_KTM_or_KMP.





Gambar 4.20 Blok Diagram Ambil Kode Tugas Mengajar

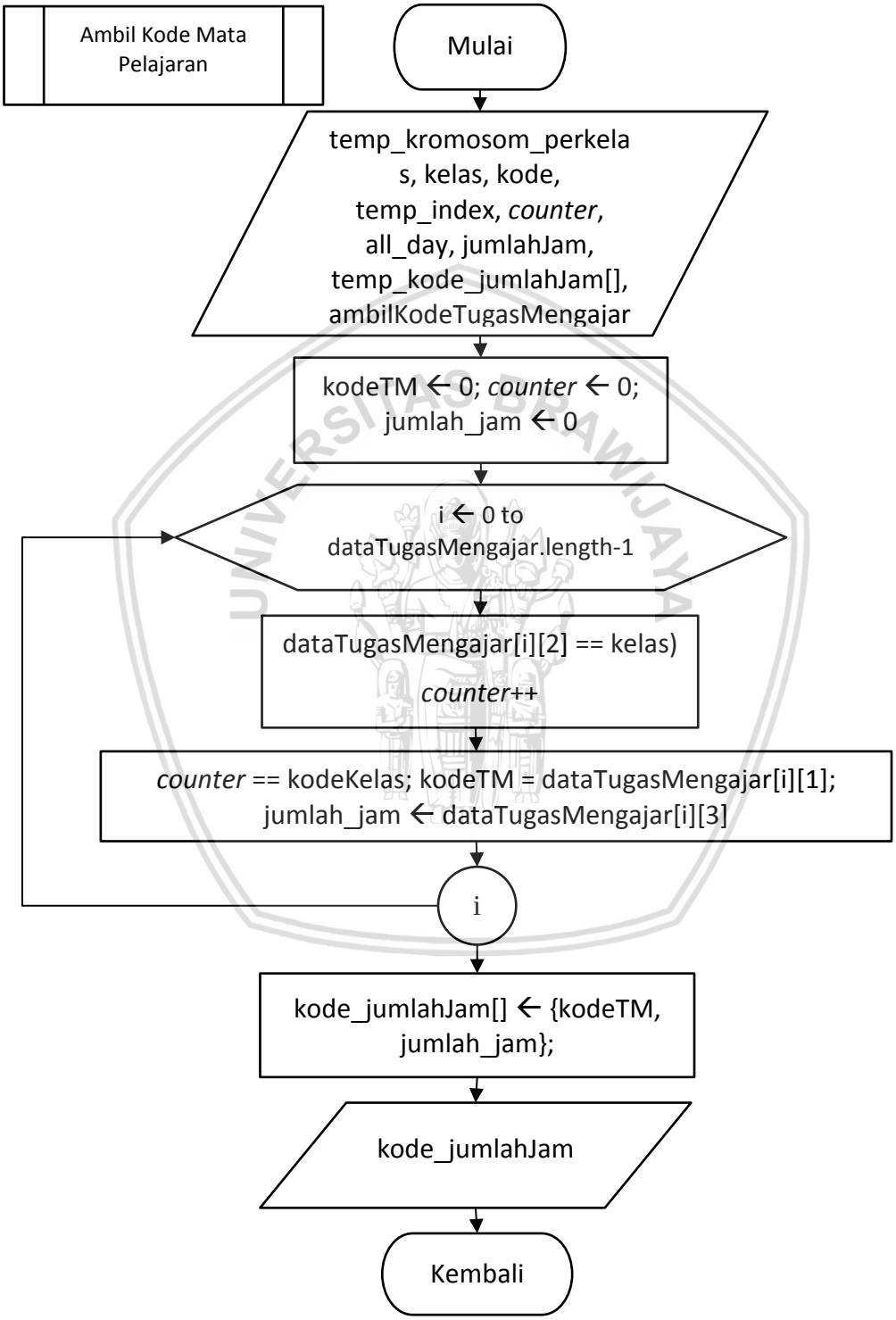
Proses yang dilakukan dalam ambil kode tugas mengajar pada Gambar 4.20 antara lain:

1. Sistem menerima masukan berupa kelas, kodekelas, kodeTM, *counter*, jumlah_jam, dataTugasMengajar, jumlah_jam, dan kode_jumlahJam.
2. Inisialisasi awal variabel kodeTM, *counter*, jumlah_jam.
3. Perulangan variabel *i* sebanyak kurang dari panjang dataTugasMengajar yang berisi apabila dataTugasMengajar[*i*][2] == kelas maka *increment* variabel *counter*.
4. Selanjutnya jika *counter* == kodeKelas, makanilai kodeTM sebanyak *i* dan jumlah_jam sebanyak dataTugasMengajar[*i*][3].
5. Inisialisasi dan pengembalian nilai variabel kode_jumlahjam[].
6. Sistem mengeluarkan hasil kode tugas mengajar.

Proses yang dilakukan dalam ambil kode mata pelajaran pada Gambar 4.21 antara lain:

1. Sistem menerima masukan berupa kelas, kodekelas, kodeTM, *counter*, jumlah_jam, dataTugasMengajar, jumlah_jam, dan kode_jumlahJam.
2. Inisialisasi awal variabel kodeTM, *counter*, jumlah_jam.
3. Perulangan variabel *i* sebanyak kurang dari panjang dataTugasMengajar yang berisi apabila dataTugasMengajar[*i*][2] == kelas maka *increment* variabel *counter*.

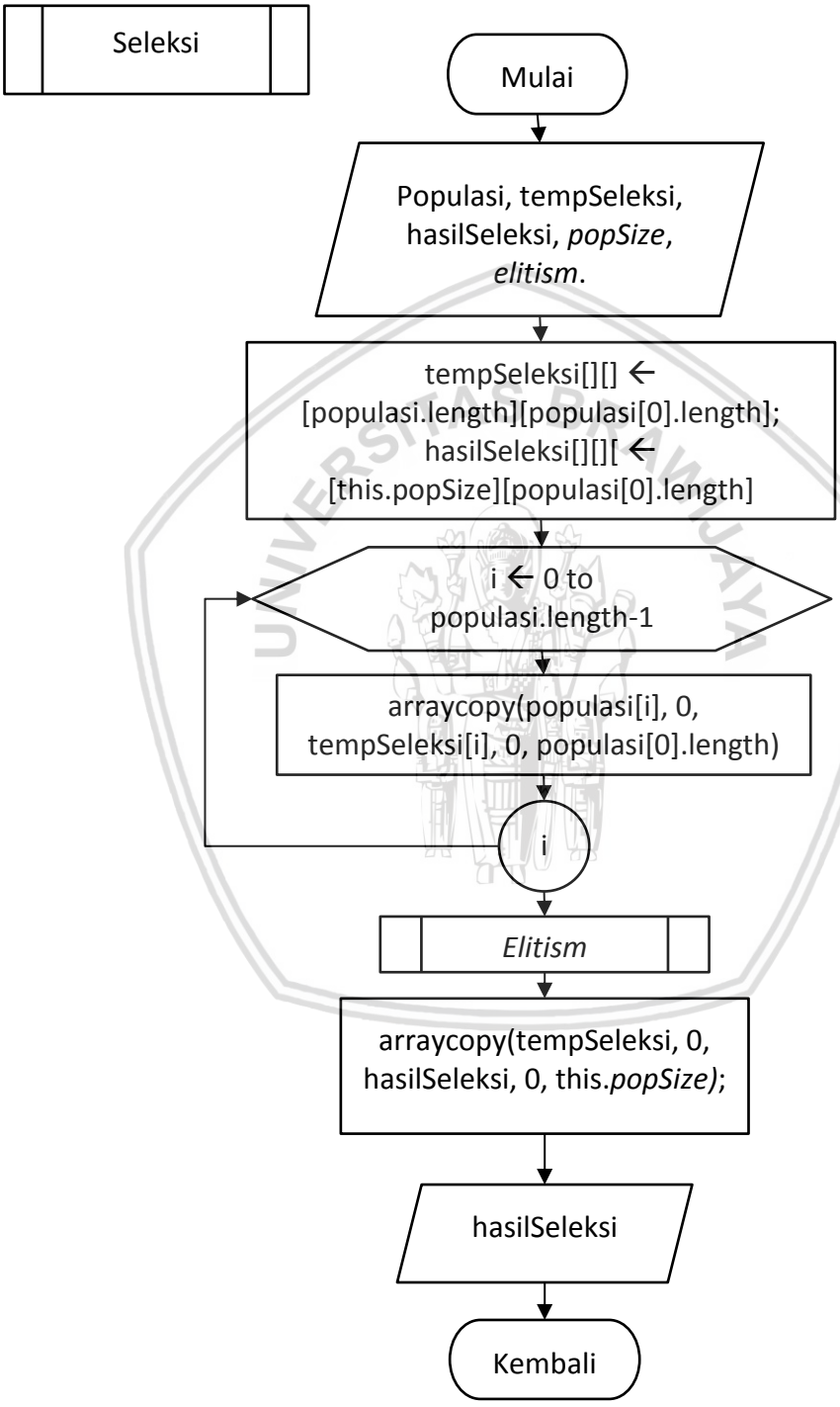
4. Selanjutnya jika *counter* == kodeKelas, makanilai kodeTM sebanyak dataTugasMengajar[*i*][1] dan jumlah_jam sebanyak dataTugasMengajar[*i*][3].
5. Inisialisasi dan pengembalian nilai variabel kode_jumlahjam[].
6. Sistem mengeluarkan hasil kode tugas mengajar.



Gambar 4.21 Blok Diagram Ambil Kode Mata Pelajaran

4.2.10 Seleksi

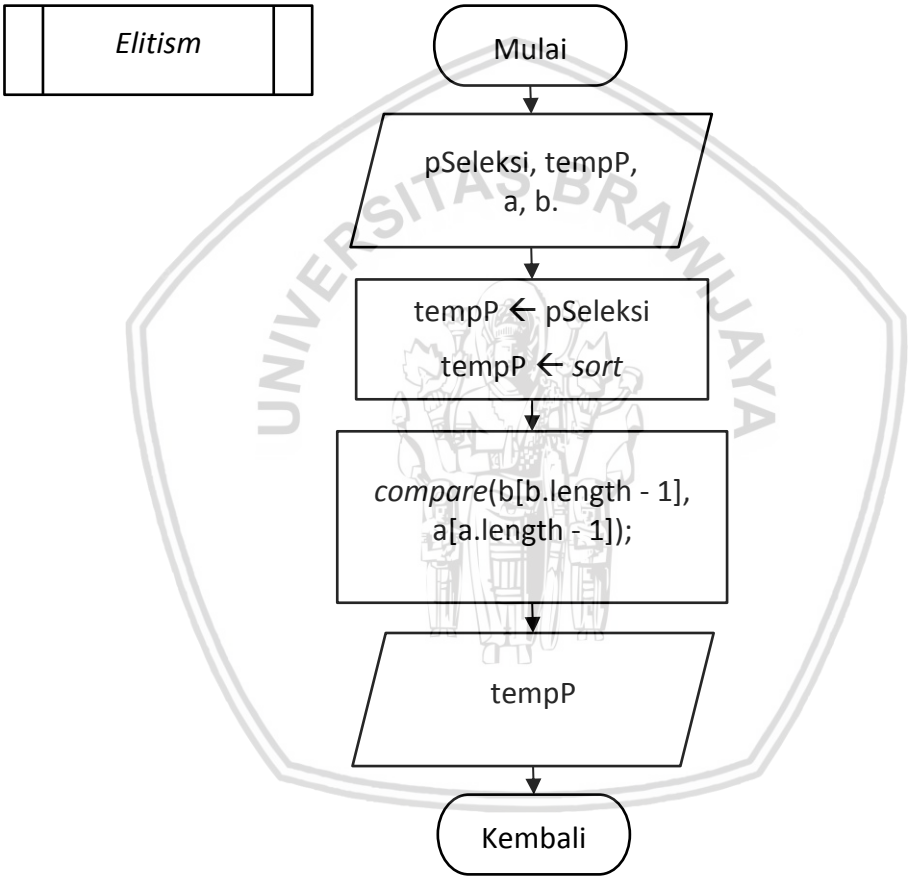
Proses seleksi dilakukan untuk mengambil individu dengan nilai *fitness* terbaik dalam populasi sebagai individu yang terpilih untuk generasi berikutnya. *Elitism selection* merupakan salah satu metode seleksi yang akan digunakan pada penelitian ini. Blok diagram seleksi dapat dilihat pada Gambar 4.13.



Gambar 4.22 Blok Diagram Seleksi

Proses yang dilakukan dalam Seleksi pada Gambar 4.22 antara lain:

1. Sistem menerima masukan berupa populasi, tempSeleksi, hasilSeleksi, *popSize*, *elitism*.
2. Inisialisasi awal variabel tempSeleksi[][] dan hasilSeleksi[][].
3. Perulangan variabel *i* sebanyak kurang dari panjang populasi untuk meng-*copy* array.
4. Memanggil method *elitism*.
5. Meng-*copy* hasil *elitism* dan megembalikan nilai variabel hasilSeleksi.
6. Sistem mengeluarkan hasil seleksi.



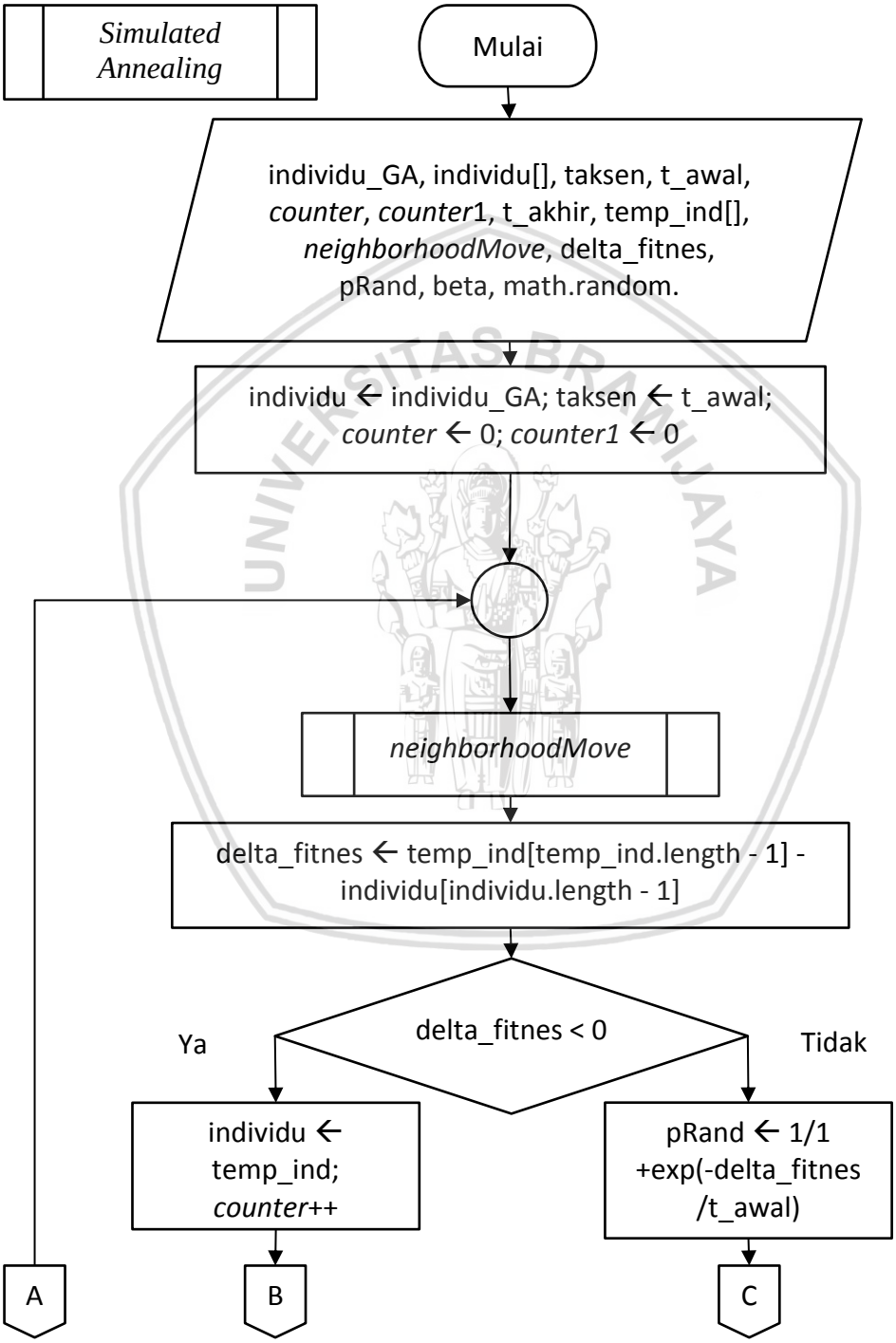
Gambar 4.23 Blok Diagram *Elitism*

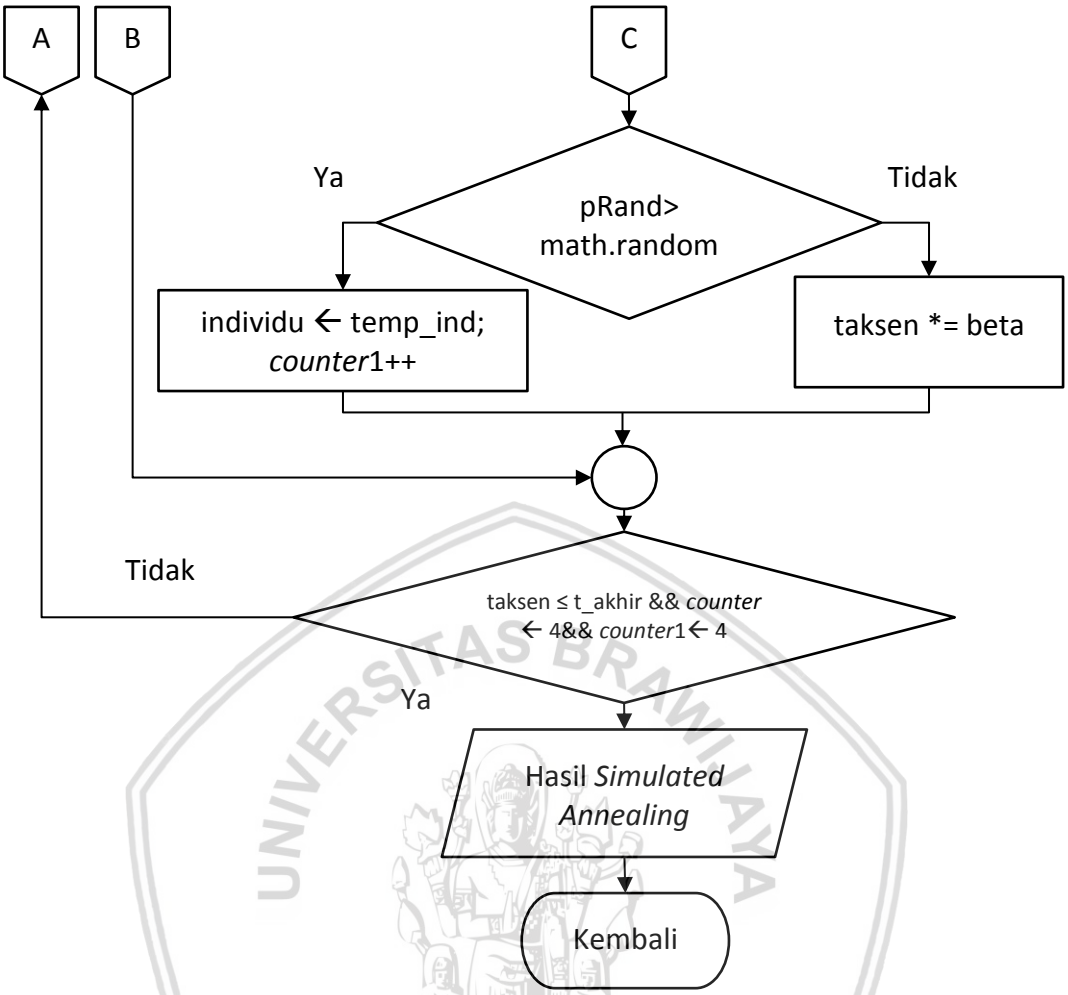
Proses yang dilakukan dalam *Elitism* pada Gambar 4.23 antara lain:

1. Sistem menerima masukan berupa pSeleksi, tempP, a, dan b.
2. Inisialisasi variabel tempP lalu di *sort*.
3. Membandingkan antara 2 buah individu, dan mengembalikan nilai variabel tempP.
4. Sistem mengeluarkan hasil *elitism*.

4.2.11 Simulated Annealing

Individu terbaik yang dihasilkan dari algoritme genetika selanjutnya akan dilakukan proses perhitungan dengan menggunakan *simulated annealing.* Beberapa proses utama dari *simulated annealing* antara lain *neighborhood move*, evaluasi, dan *probability acceptance*. Blok diagram *simulated annealing* dapat dilihat pada Gambar 4.14.



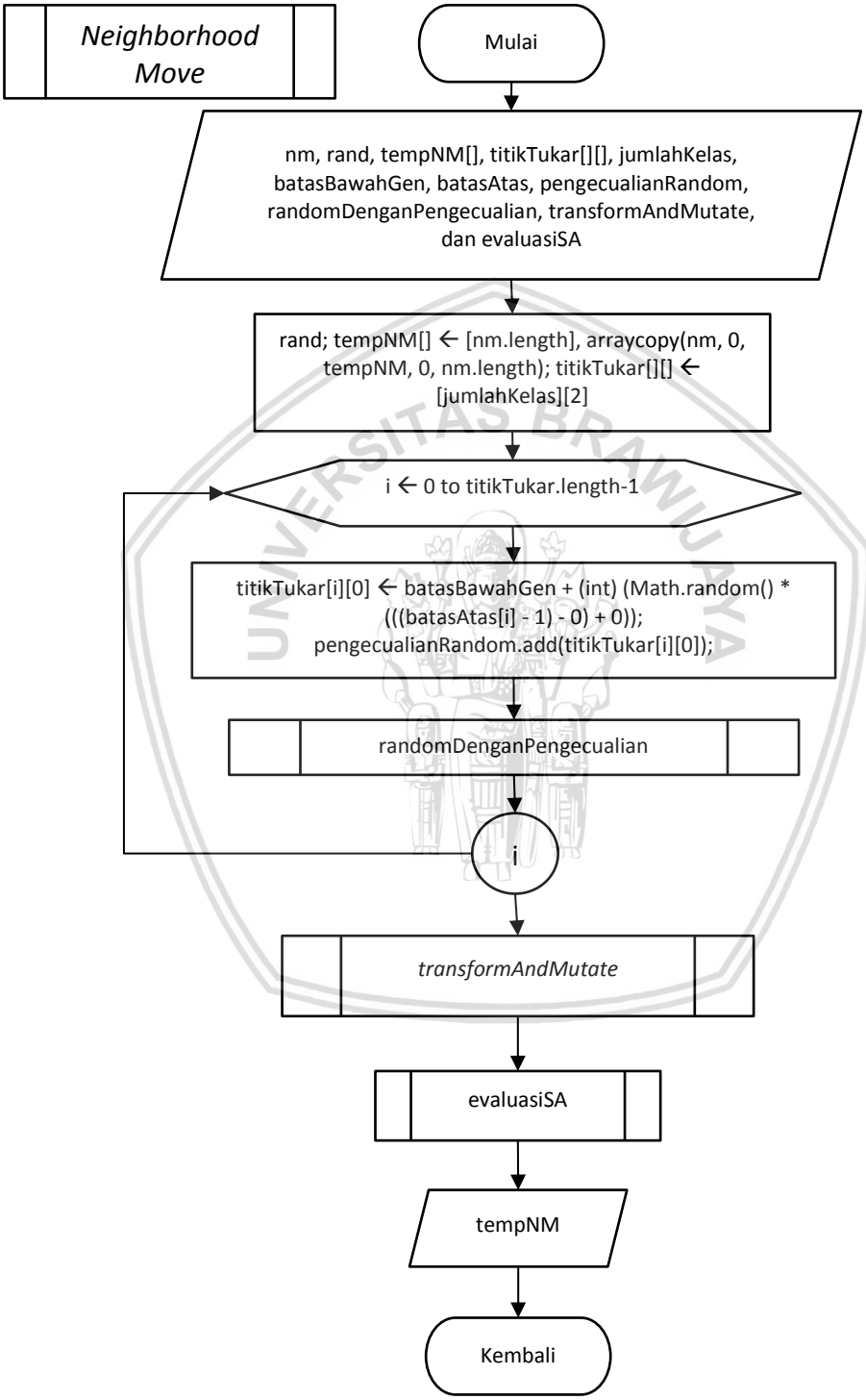


Gambar 4.24 Blok Diagram *Simulated Annealing*

Proses yang dilakukan dalam *simulated annealing* pada Gambar 4.24 antara lain:

1. Sistem menerima masukan berupa Individu_GA, individu[], taksen, t_awal, *counter*, *counter*1, t_akhir, temp_ind[], *neighborhoodMove*, delta_fitnes, pRand, beta, math.*random*.
2. Inisialisasi awal variabel individu, lalu meng-*copy* array dari individu_GA, inisialisasi variabel taksen, *counter*, dan *counter*1.
3. Masuk ada method *neighborhoodMove*.
4. Inisialisasi variabel temp_ind[] dan delta_fitnes. Apabila delta_fitnes kurang dari nol maka akan ada perbaikan individu. Jika tidak masuk ke kondisi selanjutnya.
5. Jika delta_fitnes > 0 dan pRand lebih besar dari nilai *random* maka aka nada perbaikan individu. Apabila kedua kondisi sebelumnya tidak terpenuhi maka akan dilakukan penurunan suhu.
6. Setelah dilakukan penurunan suhu, akan di cek apakah taksen ≤ t_akhir && *counter* == 4&& *counter*1== 4. Jika memenuhi maka sistem akan

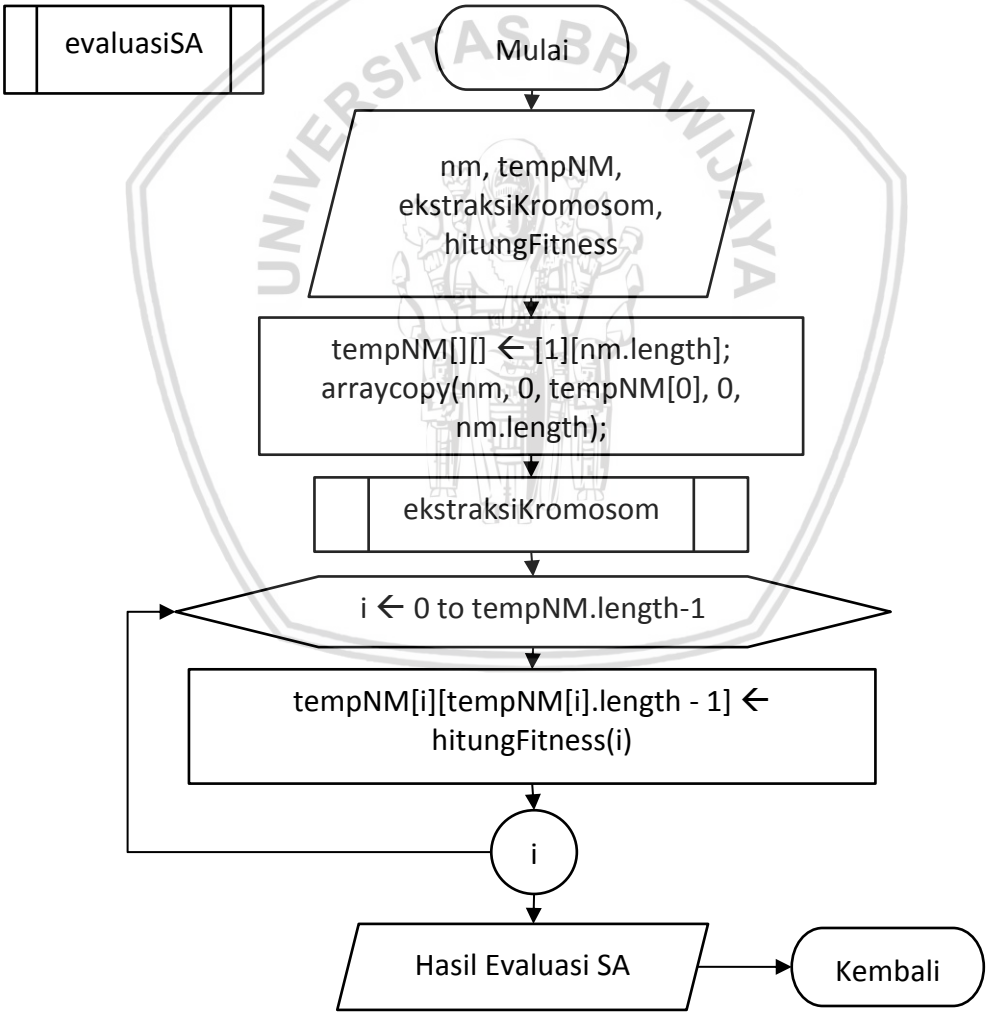
menghasilkan keluaran berupa individu hasil *simulated annealing*. Jika tidak maka akan kembali masuk pada method *neighborhoodMove*. Variable *counter* dan *counter*1 digunakan untuk membatasi perulangan apabila kondisi delta_fitnes > 0 dan pRand > *random* terpenuhi terus menerus dan untuk mempercepat waktu komputasi.



Gambar 4.25 Blok Diagram *Neighborhood Move*

Proses yang dilakukan dalam *neighborhood move* pada Gambar 4.25 antara lain:

1. Sistem menerima masukan berupa nm, rand, tempNM[], titikTukar[][], jumlahKelas, batasBawahGen, batasAtas, pengecualian*Random*, *random*DenganPengecualian, *transformAndMutate*, dan evaluasiSA.
2. Inisialisasi variabel rand, tempNM, titiktukar[][] dan meng-*copy* array.
3. Perulangan variabel *i* sebanyak kurang dari panjang titikTukar yang berisi pengisian nilai pada variabel titikTukar[*i*][0] dan menambahkannya ke dalam variabel pengecualian*Random*, serta memanggil method *random*DenganPengecualian untuk titikTukar[*i*][1].
4. Melakukan *transformAndMutate* dan evaluasiSA pada variabel tempNM.
5. Sehingga sistem akan mengeluarkan hasil dari *neighborhood move* berupa tempNM.



Gambar 4.26 Blok Diagram Evaluasi SA

Proses yang dilakukan dalam evaluasi SA pada Gambar 4.26 antara lain:

1. Sistem menerima masukan berupa Nm, tempNM, ekstraksiKromosom, hitung*Fitness*.
2. Inisialisasi variabel tempNM dan meng-*copy* array.
3. Memanggil method ekstraksiKromosom.
4. Perulangan variabel *i* sebanyak kurang dari panjang tempNM yang berisi pengisian nilai pada variabel tempNM dengan hitung*Fitness*.
5. Pengembalian nilai variabel tempNM[0] yang merupakan hasil dari proses evaluasi SA.

4.3 Perhitungan Manual Penyelesaian Optimasi Penjadwalan Mata Pelajaran Menggunakan HIbridisasi Algoritme Genetika – *Simulated Annealing* (GA-SA)

Terdapat beberapa proses dalam siklus hibridisasi algoritme GA-SA, antara lain inisialisasi kromosom, reproduksi, evaluasi, seleksi, dan *simulated annealing*. Pada perhitungan manual ini akan difokuskan pada perhitungan sederhana yang menggunakan data penjadwalan mata pelajaran pada hari senin dan selasa di kelas X IPA 1, XI IPA 1, dan XII IPA 1. Karena data penjadwalan mata pelajaran hanya diambil pada hari senin dan selasa maka dalam perhitungan manual ini diambil data penugasan guru yang mengajar tiga kelas tersebut dengan waktu masing-masing kelas 19 jam mata pelajaran.

Sebelum inisialisasi kromosom dilakukan, dibutuhkan sebuah kode unik yang mengandung nilai kode guru, nama guru, kode mata pelajaran, nama mata pelajaran, dan jumlah jam pelajaran pada mata pelajaran yang diampu. Kode tersebut didapat dari data pembagian tugas pada Tabel 4.4 yang terletak pada sub bab 4.1.

Akan tetapi data tersebut belum mempunyai kode unik untuk merepresentasikannya, maka dari itu data perlu di ekstrak menjadi data kode tugas mengajar seperti pada Tabel 4.5 yang terletak pada sub bab 4.1.

4.3.1 Inisialisasi Kromosom

Kromosom dibentuk berdasarkan gen penyusunnya, oleh sebab itu sebelum kromosom terbentuk, gen harus dibangun terlebih dahulu. Gen yang didapat berasal dari urutan tiap kelas pada masing-masing tingkatan. Panjang kromosom didapat dari jumlah keseluruhan, hasil pengurangan antara jumlah jam pelajaran dari hari senin-selasa dengan total jam pada masing-masing tingkatan kelas (X,XI,XII) dari hari senin-selasa dan ditambah dengan slot kosong masing-masing tingkatan kelas (X,XI,XII).

Dalam membentuk kromosom akan diambil data sebanyak 25 data dari kode tugas mengajar pada Tabel 4.5 yang akan diisi pada setiap gen pembentuk kromosomnya. Data yang diambil hanya 2 hari kegiatan mengajar, sehingga setiap kelas hanya membutuhkan 19 jam mata pelajaran dengan 9 jam ada hari

senin dan 10 jam pada hari selasa. Kromosom diinisialisasikan menggunakan array yang terdapat indeks dan *value*, yang mana *value* dari masing-masing indeks merupakan representasi dari gen. Contoh pembentukan kromosom dapat dilihat pada Tabel 4.6.

Tabel 4.6 Contoh Kromosom P1

P1	**X IPA 1**	2	5	3	7	8	9	4	0	1	6
	XI IPA 1	7	3	5	6	8	0	1	4	2	
	XII IPA 1	2	4	1	0	5	8	6	7	3	

Setiap gen pada kromosom berisi urutan setiap kelas pada masing-masing tingkatan dengan mengisikan angka nol pada slot yang kosong dan setiap gen tidak boleh memiliki nilai yang sama dalam satu kromosom kecuali angka nol. Contohnya gen pertama pada kromosom P1 berisi angka dua berarti kelas X IPA 1 pada urutan kedua berisi informasi mengenai tugas mengajar guru yang bernama Endang Megawati, S.Pd., M.Pd.I yang mengajar mata pelajaran Biologi pada kelas X IPA 1 dengan jumlah jam sebanyak 1 jam mata pelajaran, dan begitu seterusnya. Gen yang sudah dibentuk akan dimasukkan ke dalam *frame* jadwal seperti pada Tabel 4.6 dengan melakukan *scanning* pada setiap gen dalam kromosom P1 .

Setelah kromosom terbentuk maka akan dibangun populasi awal atau *popSize* sebagai batas jumlah populasi pada setiap generasi (iterasi). Pada penelitian ini akan menggunakan *popSize* sebanyak 3 individu dengan membangun kromosom secara *random*. Inisialisasi kromosom pada populasi awal dapat dilihat pada Tabel 4.7 sampai dengan Tabel 4.9.

Tabel 4.7 Kromosom P1

P1	**X IPA 1**	2	5	3	7	8	9	4	0	1	6
	XI IPA 1	7	3	5	6	8	0	1	4	2	
	XII IPA 1	2	4	1	0	5	8	6	7	3	

Tabel 4.8 Kromsom P2

P2	**X IPA 1**	1	6	0	4	8	9	2	5	3	7
	XI IPA 1	8	6	0	5	3	7	2	4	1	
	XII IPA 1	1	0	5	6	8	4	3	7	2	

Tabel 4.9 Kromosom P3

P3	**X IPA 1**	3	6	8	4	1	0	9	2	5	7
	XI IPA 1	5	3	0	6	4	1	2	8	7	
	XII IPA 1	0	1	5	2	6	8	3	7	4	

4.3.2 Reproduksi

Selanjutnya dalam menghasilkan individu baru akan dilakukan proses reproduksi. Proses reproduksi yang dilakukan ada dua cara yaitu *crossover* dan *mutation*. Metode *crossover* yang digunakan adalah *one-cut point crossover*, sedangkan metode mutasi yang digunakan adalah *reciprocal exchange mutation*.

4.3.2.1 Crossover

Proses *crossover* dilakukan dengan mengkawin silangkan 2 buah individu yang diambil secara *random* sebagai *parent* dalam menghasilkan keturunan. Metode yang digunakan adalah *one-cut point crossover* yaitu menukarkan susunan kromosom dengan satu titik potong yang mana menggabungkan beberapa gen dari induk pertama dan kedua.

Jumlah keturunan yang dihasilkan, ditentukan dari nilai *cr* (*crossover rate*) yang dibentuk antara 0 sampai 1 dengan menggunakan Persamaan (4.1).

$$child\_crossover = ceil\ (cr * popSize) \quad (4.1)$$

Keterangan:

child_crossover = Jumlah *child* yang dihasilkan dari proses *crossover*

cr = *Crossover rate*

popSize = Jumlah individu setiap generasi

ceil = Fungsi pembulatan ke atas

Dalam perhitungan manual ini akan menggunakan nilai *cr* sebesar 0,6 sehingga dapat ditentukan berapa jumlah keturunan yang akan dihasilkan.

$$child\_crossover = 0{,}6 * 3 = 1{,}8$$

Hasil yang didapat sebesar 1,8 maka akan dilakukan pembulatan keatas menjadi 2, jadi keturunan yang dihasilkan pada proses *crossover* ini adalah 2 keturunan. Proses *crossover* dilakukan dengan cara menentukan titik potong pada induk pertama yang mana dari gen pertama pada induk pertama sampai dengan batas titik potong akan menjadi bagian gen untuk keturunan. Lalu untuk bagian gen yang lain diambil dari induk kedua dengan mencocokkan terlebih dahulu gen mana saja yang belum ada sebelumnya pada keturunan. Proses *crossover* dapat dilihat pada Tabel 4.10.

4.3.2.2 Mutation

Proses *mutation* hanya menggunakan 1 individu sebagai induk yang dipilih secara *random*. *Mutation* dilakukan dengan mengubah susunan gen pada kromosom, hasil perubahan tersebut akan menjadi keturunannya. Jumlah keturunan yang dihasilkan, ditentukan dari nilai *mr* (*mutation rate*) yang dibentuk antara 0 sampai 1 dengan menggunakan Persamaan (4.2).

$$child\_mutation = ceil\ (mr * popSize) \quad (4.2)$$

Keterangan:

child_mutation = Jumlah *child* yang dihasilkan dari proses *mutation*

mr = *Mutation rate*

popSize = Jumlah individu setiap generasi

ceil = Fungsi pembulatan ke atas

Dalam perhitungan manual ini akan menggunakan nilai *cr* sebesar 0,3 sehingga dapat ditentukan brapa jumlah keturunan yang akan dihasilkan.

***child_mutation*= 0,3*3=0,9**

Hasil yang didapat sebesar 0,9 maka akan dilakukan pembulatan keatas menjadi 1, jadi keturunan yang dihasilkan pada proses *mutation* ini adalah 1 keturunan. Metode yang digunakan pada proses ini adalah *reciprocal exchange mutation*, yaitu dengan menukan posisi 2 buah gen dalam kromosom secara *random*. Dua gen yang ditukar didefinisikan dengan variabel *x1* dan *x2*. Proses *mutation* dapat dilihat pada Tabel 4.11.

Tabel 4.10 Proses *Crossover*

P1	**X IPA 1**	2	5	3	7	8	9	4	0	1	6
	XI IPA 1	7	3	5	6	8	0	1	4	2	
	XII IPA 1	2	4	1	0	5	8	6	7	3	
P2	**X IPA 1**	1	6	0	4	8	9	2	5	3	7
	XI IPA 1	8	6	0	5	3	7	2	4	1	
	XII IPA 1	1	0	5	6	8	4	3	7	2	
C1	**X IPA 1**	2	5	3	7	1	6	0	4	8	9
	XI IPA 1	7	3	5	6	8	0	2	4	1	
	XII IPA 1	2	4	1	0	5	6	8	3	7	
P1	**X IPA 1**	2	5	3	7	8	9	4	0	1	6
	XI IPA 1	7	3	5	6	8	0	1	4	2	
	XII IPA 1	2	4	1	0	5	8	6	7	3	
P2	**X IPA 1**	1	6	0	4	8	9	2	5	3	7
	XI IPA 1	8	6	0	5	3	7	2	4	1	
	XII IPA 1	1	0	5	6	8	4	3	7	2	
C2	**X IPA 1**	1	6	0	4	2	5	3	7	8	9
	XI IPA 1	8	6	0	5	3	7	1	4	2	
	XII IPA 1	1	0	5	6	2	4	8	7	3	

Keterangan:

: Angka-angka yang berada sebelum titik potong

: Angka-angka yang berada setelah titik potong dan angka-angka yang tidak ada pada sebelum titik potong

Tabel 4.11 Proses *Mutation*

P3	**X IPA 1**	3	6	8	4	1	0	9	2	5	7
	XI IPA 1	5	3	0	6	4	1	2	8	7	
	XII IPA 1	0	1	5	2	6	8	3	7	4	

C3	X IPA 1	3	9	8	4	1	0	6	2	5	7
	XI IPA 1	1	3	0	6	4	5	2	8	7	
	XII IPA 1	0	1	4	2	6	8	3	7	5	

Keterangan:

: Variabel *x1*

: Variabel *x2*

4.3.3 Pengecekan *Constraint* Ke-1

Sebelum dilakukan pengecekan *constraint* maka terlebih dahulu dilakukan pembobotan pada setiap *constraint*. Terdapat 2 jenis *constraint* yang didefinisikan, yaitu *hard constraint* dan *soft constraint*. Tentu saja nilai bobot pada masing masing jenis constraint ini berbeda. Untuk jenis *hard constraint* diberi nilai bobot sebesar 0,005 sedangkan untuk jenis *soft constraint* diberi nilai bobot sebesar 0,001. Penjelasan mengenai pembobotan pada setiap *constraint* dapat dilihat pada Tabel 4.12.

Tabel 4.12 Pembobotan Setiap *Constraint*

NO	JENIS *CONSTRAINT*	*CONSTRAINT*	BOBOT PELANGGARAN
1	*Hard Constraint*	Guru yang mengajar tidak boleh mengajar lebih dari satu kelas atau mengajar di lain kelas pada waktu yang sama (bentrok)	0.005
2	*Soft Constraint*	Guru yang mengajar tidak boleh mengajar lebih dari empat jam pelajaran dalam mata pelajaran yang sama dan di hari yang sama	0.001
3	*Soft Constraint*	Mata pelajaran Penjaskes OR hanya boleh berada pada jam pelajaran ke 1-4	0.001
4	*Soft Constraint*	Dalam satu mata pelajaran yang sama pada jam kedua dan selanjutnya (sejumlah ketentuan jumlah jamnya) tidak boleh beda hari	0.001

Bobot nilai pada *hard constraint* lebih besar dibanding dengan bobot nilai *soft constraint* karena jika *hard constraint* dilanggar maka akan berakibat fatal pada kegiatan belajar mengajar.

Untuk pengecekan pada *constraint* ke-1, pelanggaran akan terjadi apabila terdapat guru yang mengajar lebih dari satu kelas pada waktu yang sama (bentrok). Dalam penghitungannya dapat dilakukan dengan menyusun jadwal berdasarkan kode tugas mengajar. Jadwal tersebut dapat dilihat pada Tabel 4.13.

Berdasarkan Tabel 4.13 terdapat bentok, sehingga nilai pelanggaran pada *constraint* ke-1 sebesar:

f(1) = jumlah pelanggaran * bobot *constraint* ke-1

f(1) = 2 * 0,005 = 0,01

Tabel 4.13 Pengecekan *Constraint* ke-1 Berdasarkan Kode Tugas Mengajar

HARI	SENIN									SELASA								
KELAS / JAM	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
X IPA 1	6	6	13	13	7	17	17	19	19	35	35	35	12	0	0	0	0	15	15
XI IPA 1	38	38	10	10	33	33	36	36	39	39	0	7	7	27	27	9	9	9	9
XIII IPA 1	1	1	1	1	14	14	0	0	0	25	25	40	40	26	26	27	27	9	9
PELANGGARAN	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	Y

Keterangan:

: Pelanggaran *constraint* ke-1

: Jam kosong

4.3.4 Pengecekan *Constraint* ke-2

Untuk pengecekan pada *constraint* ke-2, pelanggaran akan terjadi apabila mata pelajaran penjaskes berada di atas jam pelajaran ke-4. Dalam penghitungannya dapat dilakukan dengan menyusun jadwal berdasarkan kode mata pelajaran. Jadwal tersebut dapat dilihat pada Tabel 4.14.

Tabel 4.14 Pengecekan *Constraint* ke-2 Berdasarkan Kode Mata Pelajaran

HARI	SENIN									SELASA									
KELAS / JAM	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
X IPA 1	7	7	10	10	9	2	2	14	14	17	17	17	2	0	0	0	0	11	11
XI IPA 1	11	11	1	1	14	14	13	13	0	0	0	4	4	3	3	9	9	9	9
XIII IPA 1	2	2	2	2	11	11	0	0	0	9	9	10	10	12	12	3	3	4	4
PELANGGARAN	N	N	N	N	N	N	Y	Y	Y	N	N	N	N	N	Y	Y	Y	N	N

Keterangan:

: Pelanggaran *constraint* ke-2

: Jam kosong

Berdasarkan tabel di atas terdapat pelanggaran pada jam ke-8 sampai jam ke-10 di hari senin dan jam ke-6 sampai jam ke-8 di hari selasa, sehingga nilai pelanggaran pada *constraint* ke-2 sebesar:

f(2) = jumlah pelanggaran * bobot *constraint* ke-2

f(2) = 6 * 0,001 = 0,006

4.3.5 Pengecekan *Constraint* ke-3

Untuk pengecekan pada *constraint* ke-3, pelanggaran akan terjadi apabila dalam satu kelas terdapat lebih dari 4 jam pada mata pelajaran yang sama.

Dalam penghitungannya dapat dilakukan dengan menyusun jadwal berdasarkan kode mata pelajaran. Jadwal tersebut dapat dilihat pada Tabel 4.15.

Tabel 4.15 Pengecekan *Constraint* ke-3 Berdasarkan Kode Mata Pelajaran

HARI	SENIN										
KELAS / JAM	2	3	4	5	6	7	8	9	10		Y/N
X IPA 1	7	7	10	10	9	2	2	14	14		N
XI IPA 1	11	11	1	1	14	14	13	13	0		N
XIII IPA 1	2	2	2	2	11	11	0	0	0		N
HARI	SELASA										
KELAS / JAM	1	2	3	4	5	6	7	8	9	10	Y/N
X IPA 1	17	17	17	2	0	0	0	0	11	11	N
XI IPA 1	0	0	4	4	3	3	9	9	9	9	N
XIII IPA 1	9	9	10	10	12	12	3	3	4	4	N

Keterangan:

: Pelanggaran *constraint* ke-3

: Jam kosong

Berdasarkan tabel di atas tidak terdapat pelanggaran, sehingga didapat nilai pelanggaran pada *constraint* ke-3 sebesar:

f(3) = jumlah pelanggaran * bobot *constraint* ke-3

f(3) = 0* 0,001 = 0

4.3.6 Pengecekan *Constraint* ke-4

Untuk pengecekan pada *constraint* ke-4, pelanggaran akan terjadi apabila dalam satu mata pelajaran yang sama pada jam kedua dan selanjutnya (sejumlah ketentuan) berada pada hari yang berbeda. Dalam penghitungannya dapat dilakukan dengan menyusun jadwal berdasarkan kode mata pelajaran. Jadwal tersebut dapat dilihat pada Tabel 4.16.

Tabel 4.16 Pengecekan *Constraint* ke-4 Berdasarkan Kode Mata Pelajaran

HARI	SENIN									SELASA									Y/N
KELAS / JAM	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	
X IPA 1	7	7	10	10	9	2	2	14	14	17	17	17	2	0	0	0	0	11	11	N
XI IPA 1	11	11	1	1	14	14	13	13	0	0	0	4	4	3	3	9	9	9	9	Y
XIII IPA 1	2	2	2	2	11	11	0	0	0	9	9	10	10	12	12	3	3	4	4	N

Keterangan:

: Pelanggaran *constraint* ke-4

: Jam kosong

Berdasarkan tabel di atas terdapat pelanggaran pada jam ke-10 di hari senin dan jam ke-1 di hari selasa sehingga didapat nilai pelanggaran pada *constraint* ke-4 sebesar:

f(4) = jumlah pelanggaran * bobot *constraint* ke-4

f(4) = 1 * 0,001 = 0,001

4.3.7 Perhitungan Nilai *Fitness*

Nilai *fitness* didapat dari fungsi objektif, yaitu jumlah seluruh nilai pelanggaran pada setiap kromosom, dapat diliat pada Pesamaan (2.1). Selanjutnya fungsi objektif tersebut dihitung dengan menggunakan rumus fitness pada Persamaan (2.2).

Jumlah pelanggaran *constraint* ke-1 = 2

Jumlah pelanggaran *constraint* ke-2 = 6

Jumlah pelanggaran *constraint* ke-3 = 0

Jumlah pelanggaran *constraint* ke-4 = 1

f(x) = (2*0,005)+(6*0,001)+(0*0,001)+(1*0,001)

= 0,017

fitness = 1/(1+0,017)

= 0,983

4.3.8 Evaluasi

Pada proses evaluasi dilakukan penggabungan seluruh individu dalam populasi beserta keturunannya dengan nilai *fitness* yang dimiliki pada setiap individu. Individu dengan nilai tertinggi memiliki kemungkinan besar untuk dapat masuk pada generasi berikutnya. Gabungan seluruh individu berserta nilai *fitness*-nya dapat dilihat pada Tabel 4.17.

Tabel 4.17 Hasil Evaluasi

Individu	Kromosom	Jumlah dan Total Pelanggaran (*f(x)*)	Nilai *Fitness*
P1	[2,5,3,7,8,9,4,0,1,6 7,3,5,6,8,0,1,4,2 2,4,1,0,5,8,6,7,3]	(2*0,005)+(6*0,001)+(0 *0,001)+(1*0,001) =0,017	1/(1+0,017) = 0,983
P2	[1,6,0,4,8,9,2,5,3,7 8,6,0,5,3,7,2,4,1 1,0,5,6,8,4,3,7,2]	(4*0,005)+(0*0,001)+(0 *0,001)+(0*0,001) =0,02	1/(1+0,002) = 0,9804
P3	[3,6,8,4,1,0,9,2,5,7 5,3,0,6,4,1,2,8,7 0,1,5,2,6,8,3,7,4]	(2*0,005)+(5*0,001)+(0 *0,001)+(0*0,001) =0,015	1/(1+0,015) = 0,9852
C1	[2,5,3,7,1,6,0,4,8,9	(1*0,005)+(3*0,001)+(0	1/(1+0,01)

	7,3,5,6,8,0,2,4,1 2,4,1,0,5,6,8,3,7]	*0,001)+(2*0,001) =0,01	= 0,9901
C2	[1,6,0,4,2,5,3,7,8,9 8,6,0,5,3,7,1,4,2, 1,0,5,6,2,4,8,7,3]	(4*0,005)+(0*0,001)+(0 *0,001)+(1*0,001) =0,021	1/(1+0,021) = 0,9794
C3	[3,9,8,4,1,0,6,2,5,7 1,3,0,6,4,5,2,8,7 0,1,4,2,6,8,3,7,5]	(1*0,005)+(4*0,001)+(0 *0,001)+(1*0,001) =0,01	1/(1+0,01) = 0,9901

4.3.9 Seleksi

Proses seleksi pada penelitian ini meggunakan metode *elitism selection*, yang mana akan dilakukan pemilihan terhadap individu yang mempunyai nilai *fitness* tertinggi sejumlah populasi awal berdasarkan hasil evaluasi sebelumnya. Individu hasil seleksi ini akan gunakan sebagai individu pada populasi di generasi selanjutnya. Hasil dari proses seleksi dapat dilihat pada Tabel 4.18.

Tabel 4.18 Hasil Seleksi

Individu	Kromosom	Jumlah dan Total Pelanggaran (*f(x)*)	Nilai *Fitness*
C3	[3,9,8,4,1,0,6,2,5,7 1,3,0,6,4,5,2,8,7 0,1,4,2,6,8,3,7,5]	(1*0,005)+(4*0,001)+(0 *0,001)+(1*0,001) =0,01	1/(1+0,01) = 0,9901
C1	[2,5,3,7,1,6,0,4,8,9 7,3,5,6,8,0,2,4,1 2,4,1,0,5,6,8,3,7]	(1*0,005)+(3*0,001)+(0 *0,001)+(2*0,001) =0,01	1/(1+0,01) = 0,9901
P3	[3,6,8,4,1,0,9,2,5,7 5,3,0,6,4,1,2,8,7 0,1,5,2,6,8,3,7,4]	(2*0,005)+(5*0,001)+(0 *0,001)+(0*0,001) =0,015	1/(1+0,015) = 0,9852

4.3.10 Simulated Annealing

Individu terbaik yang dihasilkan dari algoritme genetika selanjutnya akan dilakukan proses perhitungan dengan menggunakan *simulated annealing*. Individu terbaik yang didapat adalah C3 dengan nilai *fitness* tertinggi. Parameter yang digunakan adalah *Tc* (suhu awal) sebesar 15, *Ta* (suhu akhir) sebesar 10, *β* (koefisien penurunan suhu) sebesar 0,5, dan *r* (bilangan *random*) sebesar 0,6.

Proses *simulated annealing* yang pertama adalah *neighborhood move*, yang mana proses tersebut dilakukan penukaran dua nilai gen yang berbeda untuk mendapatkan individu baru (C4) dapat dilihat pada Tabel 4.19.

Tabel 4.19 *Neighborhood Move*

C3	**X IPA 1**	3	9	8	4	1	0	6	2	5	7
	XI IPA 1	1	3	0	6	4	5	2	8	7	
	XII IPA 1	0	1	4	2	6	8	3	7	5	
C4	**X IPA 1**	2	9	8	4	1	0	6	3	5	7
	XI IPA 1	1	2	0	6	4	5	3	8	7	
	XII IPA 1	0	1	7	2	6	8	3	4	5	

Keterangan:

: Titik tukar pertaama

: Titik tukar kedua

Setelah itu menghitung nilai *fitness* dari inidividu C4 untuk dilakukan evaluasi, evaluasi nilai *fitness* dapat dilihat pada Tabel 4.22.

Tabel 4.20 Evaluasi Nilai *Fitness*

Individu	Kromosom	Jumlah dan Total Pelanggaran (*f(x)*)	Nilai *Fitness*
C3	[3,9,8,4,1,0,6,2,5,7 1,3,0,6,4,5,2,8,7 0,1,4,2,6,8,3,7,5]	(1*0,005)+(4*0,001)+(0 *0,001)+(1*0,001) =0,01	1/(1+0,01) = 0,9901
C4	[2,5,3,7,1,6,0,4,8,9 7,3,5,6,8,0,2,4,1 2,4,1,0,5,6,8,3,7]	(0*0,005)+(3*0,001)+(0 *0,001)+(1*0,001) =0,004	1/(1+0,004) = 0,9960

Berdasarkan Persamaan (2.3) akan dilakukan perhitungan selisih nilai *fitness* antara individu P3 dan C4 seperti perhitungan dibawah ini:

$$\Delta f = 0{,}9960 - 0{,}9901 = 0{,}0059$$

Apabila Δf tidak kurang dari 0, maka akan dilakukan perhitungan pembanding nilai random berdasarkan Persamaan (2.4) seperti perhitungan dibawah ini:

$$pRand = 1/(1+\exp(-0{,}0059/15) = 0{,}5001$$

Karena nilai *pRand* tidak lebih besar dari nilai *random* sebesar 0,6, maka akan dilakukan penurunan suhu berdasarkan Persamaan (2.5) seperti perhitungan dibawah ini:

$$Tc^{(new)} = 0{,}5*15 = 7{,}5$$

Hasil penurunan suhu $Tc^{(new)}$ kurang dari suhu akhir (*Ta*), maka individu C3 terpilih untuk iterasi GA-SA selanjutnya.

4.4 Perancangan Skenario Pengujian

Perancangan pengujian dilakukan pada pengujian terhadap parameter-parameter yang ada di hibridisasi algoritme GA-SA. Skenario pengujian paramere-parameter tersebut, antara lain:

1. Pengujian generasi.
2. Pengujian *popSize*.
3. Pengujian parameter reproduksi (*crossover rate* dan *mutation rate*).
4. Pengujian algoritme (GA-SA dengan GA).
5. Pengujian konvergensi.
6. Pengujian global.

4.4.2 Pengujian Generasi

Pengujian generasi dilakukan untuk mendapatkan rata-rata nilai *fitness* terbaik dengan mengkombinasikan jumlah populasi, nilai *cr* dan *mr*. rancangan pengujian generasi dapat dilihat pada Tabel 4.23.

Tabel 4.21 Rancangan Pengujian Generasi

Uji Coba ke-i	Generasi									
	30	60	90	120	150	180	210	240	270	300
1										
2										
3										
…										
10										
Rata-Rata										

4.4.3 Pengujian *PopSize*

Pengujian *popSize* dilakukan untuk menghindari terjadinya konvergensi dini. Karena semakin besar nilai *popSize* maka akan semakin banyak keberagaram individu. Rancangan pengujian *popSize* dapat dilihat pada Tabel Tabel 4.24.

Tabel 4.22 Rancangan Pengujian *PopSize*

Uji Coba ke-i	*PopSize*									
	10	20	30	40	50	60	70	80	90	100
1										
2										
3										
…										
10										
Rata-Rata										

4.4.4 Pengujian Parameter Reproduksi

Pengujian parameter reproduksi dilakukan dengan cara merubah nilai pada *crossover rate* (*cr*) dan *mutation rate* (*mr*). Semakin besar *cr* dan *mr* maka semakin bertambah kemungkinan solusi, tetapi waktu komputasi yang diperlukan lama pula. Sedangkan dengan nilai *cr* dan *mr* yang kecil akan memakan waktu komputasi yang singkat tetapi pencrian solusi menjadi semakin sempit. Rancangan pengujian parameter reproduksi dapat dilihat pada Tabel 4.25.

Tabel 4.23 Rancangan Pengujian Parameter Reproduksi

Uji Coba ke-i	Parameter (*cr*,*mr*)								
	0.1, 0.9	0.2, 0.8	0.3, 0.7	0.4, 0.6	0.5, 0,5	0.6, 0.4	0.7, 0.3	0.8, 0.2	0.9, 0.1
1									
2									
3									
…									
10									
Rata-Rata									

4.4.5 Pengujian Algoritme

Pengujian algoritme ini dilakukan untuk mengetahui apakah hibridisasi algoritme GA-SA lebih baik dibanding dengan algoritme genetika murni.

Pengujian ini menggunakan nilai dari generasi, populasi, nilai *cr* dan *mr* yang konstan. Rancangan pengujian algoritme dapat dilihat pada Tabel 4.26.

Tabel 4.24 Rancangan Pengujian Algoritme

Uji Coba ke-i	Generasi, *popSize*, *cr*, *mr*	
	90, 270, 0.3, 0.7	
	GA	GA-SA
1		
2		
3		
…		
10		
Rata-Rata		

4.4.6 Pengujian Konvergensi

Pengujian konvergensi dilakukan untuk mengetahui pada generasi keberapa nilai *fitness* yang dihasilkan sudah tidak mengalami perbedaan yang signifikan atau sudah stabil tidak ada perubahan. Parameter yang digunakan merupakan parameter terbaik dari hasil pengujian sebelumnya. Pengujian konvergensi ini dilakukan dengan melakukan sebanyak sepuluh kali percobaan pada setiap generasi yang ditentukan.

4.4.7 Pengujian Global

Pengujian global ini dilakukan untuk membandingkan hasil akhir antara penjadwalan secara manual dengan hasil akhir penjadwalan dengan sistem. Apakah hasil penjadwalan dengan sistem lebih baik daripada penjadwalan secara manual ataupun sebaliknya. Parameter yang digunakan untuk penjadwalan dengan sistem adalah parameter-parameter terbaik dari hasil pengujian sebelumnya.

4.5 Perancangan Antarmuka

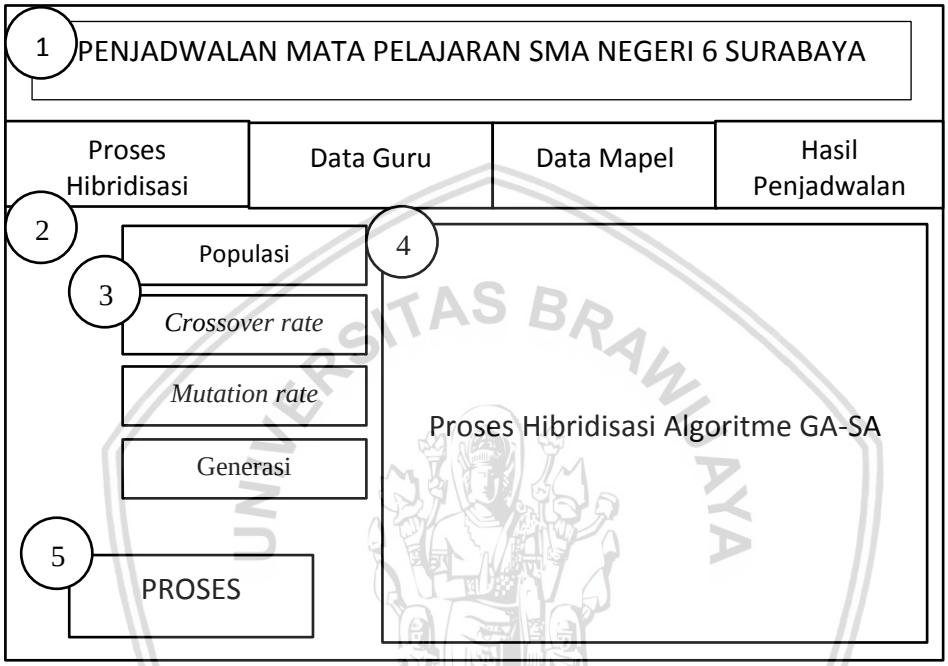
Perancangan antarmuka betujuan memodelkan antarmuka untuk mengimplementasikan sistem.

4.5.1 Perancangan Halaman Proses Hibridisasi Algoritma GA-SA

Pada halaman ini untuk menampilkan proses hibridisasi *Algoritme* GA-SA dengan menginputkan jumlah generasi, populasi, nilai *cr*, dan nilai *mr*. Perancangan halaman proses hibridisasi *Algoritme* dapat dilihat pada Gambar 4.27.

Keterangan Gambar 4.27:

1. Judul aplikasi.
2. Menu aplikasi.
3. Kolom input parameter.
4. Kolom proses hibridisasi *Algoritme* GA-SA.
5. Button untuk memproses inputan parameter.



Gambar 4.27 Perancangan Halaman Proses Hibridisasi Algoritma GA-SA

4.5.2 Perancangan Halaman Data Guru

Pada halaman ini untuk menampilkan data guru yang ada di SMA Negeri 6 Surabaya. Perancangan halaman data guru dapat dilihat pada Gambar 4.28.

Keterangan Gambar 4.28:

1. Judul aplikasi.
2. Menu aplikasi.
3. Tabel daftar guru.

4.5.3 Perancangan Halaman Data Mata Pelajaran

Pada halaman ini untuk menampilkan data mata pelajaran yang ada di SMA Negeri 6 Surabaya. Perancangan halaman data mata pelajaran dapat dilihat pada Gambar 4.29.

Keterangan Gambar 4.29:

1. Judul aplikasi.

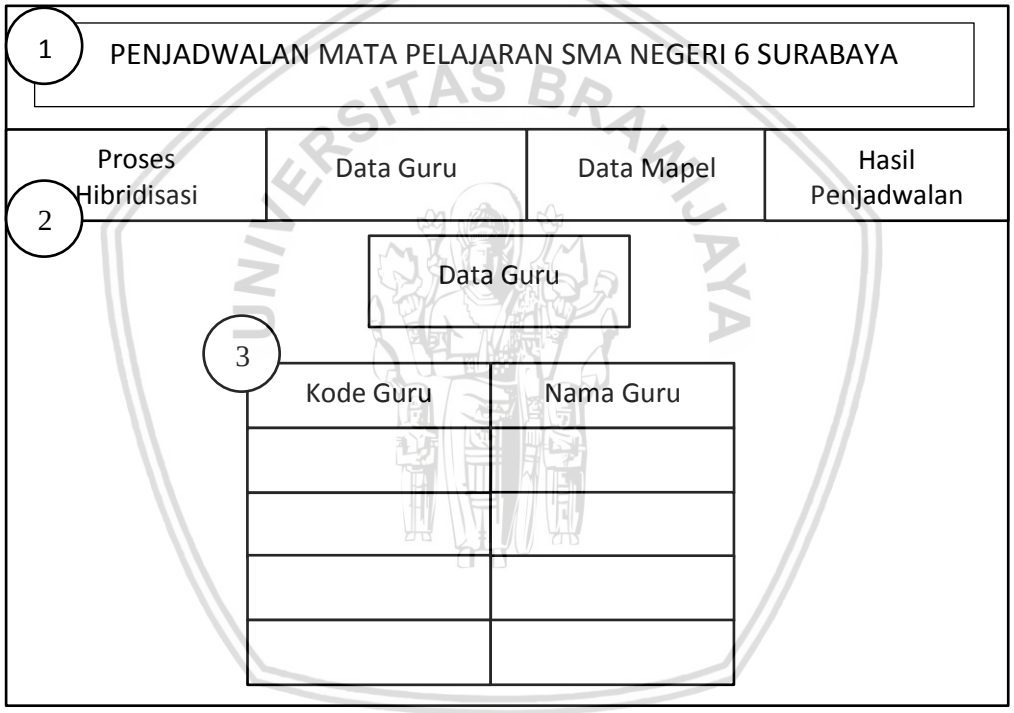
2. Menu aplikasi.
3. Tabel daftar mata pelajaran.

4.5.4 Perancangan Halaman Hasil Penjadwalan

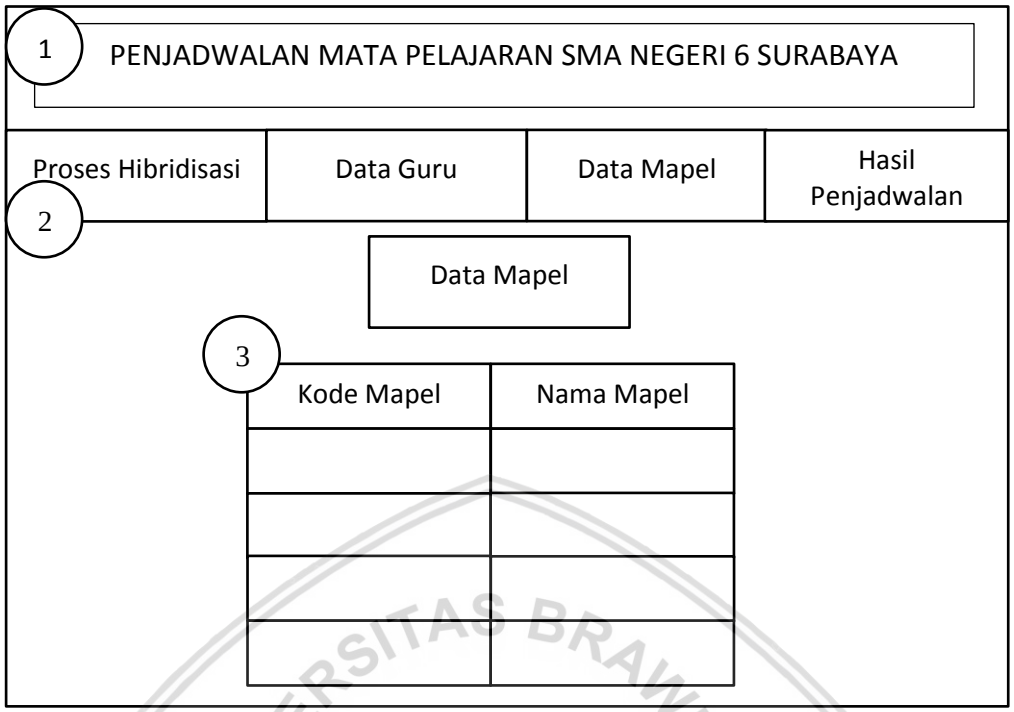
Pada halaman ini untuk menampilkan hasil penjadwalan dari proses hibridisasi *Algoritme* GA-SA yang ada di SMA Negeri 6 Surabaya. Perancangan halaman hasil penjadwalan dapat dilihat pada Gambar 4.30.

Keterangan Gambar 4.30:

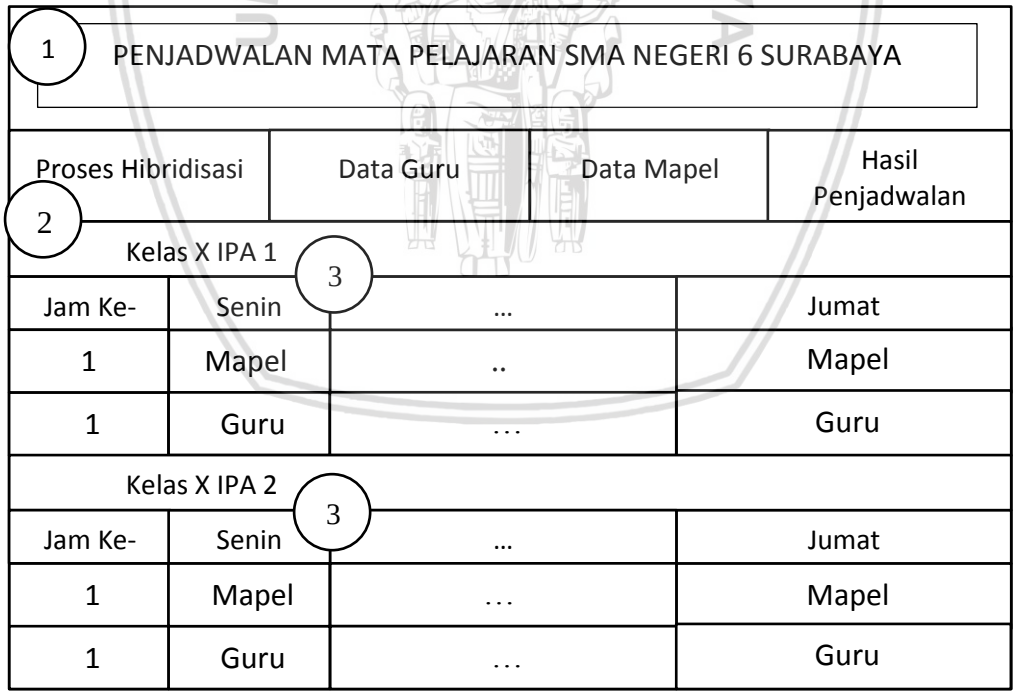
1. Judul aplikasi.
2. Menu aplikasi.
3. Jadwal mata pelajaran berdasarkan kelas.



Gambar 4.28 Perancangan Halaman Data Guru



Gambar 4.29 Perancangan Halaman Data Mata Pelajaran



Gambar 4.30 Perancangan Halaman Hasil Penjadwalan

BAB 5 IMPLEMENTASI

Bab impelemntasi membahas mengenai kode program sistem yang akan dibuat berdasarkan bab sebelumnya, yaitu analisis kebutuhan dan perancangan sistem. Pada bab ini akan dijelaskan mengenai kode program penerapan *Algoritme* genetika dan *simulated annealing* pada studi kasus optimasi penjadwalan mata pelajaran di SMA Negeri 6 Surabaya beserta perjelasan kode program untuk antarmukanya.

5.1 Implemetasi Hibridisasi Algoritma Genetika dan *Simulated Annealing*

Pada implementasi ini akan dijelaskan mengenai kode program yang menerapkan hibridisasi *Algoritme* genetika dan *simulated annealing* pada studi kasus optimasi penjadwalan mata pelajaran. Beberapa kode program yang akan dijelaskan antara lain, inisialisasi kromosom, *crossover*, *mutation*, pembentukan matriks jadwal, pengecekan *constraint*, evaluasi, seleksi, dan *simulated annealing*.

5.1.1 Implementasi Inisialisasi Kromosom

Proses inisialisasi kromosom dimulai dengan membaca data yang telah diolah menjadi kode dalam bentuk file *note*. Kode-kode tersebut direpresentasikan menjadi sebuah individu. Setiap individu terdiri dari beberapa gen, dan setiap gen dalam individu membawa sejumlah informasi. Informasi tersebut terdiri dari nama guru yang mengajar mata pelajaran tertentu pada kelas tertentu dengan jumlah jam yang telah ditetapkan. Implementasi inisialisasi kromosom tersebut dapat dilihat lebih jelas pada Kode Program 5.1.

```
1.  double[][] inisialisasiKromosom1() {
2.  Random rand = new Random();
3.  double[][] populasi = new double[this.popSize][];
4.
5.  for (int i = 0; i < this.popSize; i++) {
6.   double[] temp_ind = new double[this.jumlahKromosom + 1];
7.   int temp_index = 0;
8.   ArrayList<Integer> jamKosong = new ArrayList<>();
9.
10.   for (int j = 0; j < this.jumlahKelas; j++) {
11.    int[] temp_kelas = new int[this.panjangGenKelas[j]];
12.    ArrayList<Integer> temp_jamKosong = new ArrayList<>();
13.
14.    for (int k = 0; k < this.slotKosong[j]; k++) {
15.     int temp;
16.     if (temp_jamKosong.isEmpty()) {
17.      temp = this.batasBawahGen + (int) (Math.random()
18.      * ((this.batasAtas[j] - this.batasBawahGen) + 1));
19.     } else {
20.      temp = randomDenganPengecualian(rand,
21.      this.batasBawahGen, this.batasAtas[j],
22.      temp_jamKosong);
23.     }
```

```
24.      temp_jamKosong.add(temp);
25.      Collections.sort(temp_jamKosong);
26.     }
27.
28.     ArrayList<Integer> index_tidak_jamKosong = new
29.     ArrayList<>();
30.     for (int k = 0; k < this.panjangGenKelas[j]; k++) {
31.      if (!temp_jamKosong.contains(k)) {
32.      int temp;
33.       if (index_tidak_jamKosong.isEmpty()) {
34.        temp = this.batasBawahGen + (int) (Math.random()
35.        * ((this.batasAtas[j] - this.batasBawahGen) + 1));
36.       } else {
37.        temp = randomDenganPengecualian(rand,
38.        this.batasBawahGen, this.batasAtas[j],
39.        index_tidak_jamKosong);
40.       }
41.       index_tidak_jamKosong.add(temp);
42.       Collections.sort(index_tidak_jamKosong);
43.       temp_kelas[k] = temp;
44.       temp_ind[k + temp_index] = temp;
45.      }
46.     }
47.
48.     temp_index += this.panjangGenKelas[j];
49.    }
50.
51.    populasi[i] = temp_ind;
52.   }
53.
54.   return populasi;
55. }
```

Kode Program 5.1 Implementasi Inisialisasi Kromosom

Proses yang dilakukan dalam inisialisasi kromosom pada Kode Program 5.1 antara lain:

1. Baris 2-3 untuk inisialisasi variabel rand dan populasi array 2 dimensi.
2. Baris 14-26 untuk menentukan jumlah slot kosong pada masing-masing kelas.
3. Baris 30-46 untuk menentukan panjang gen pada masing-masing kelas.
4. Baris 10-49 untuk perulangan panjang gen sebanyak jumlah kelas.
5. Baris 5-52 untuk perulangaan panjang gen sebanyak *popSize*.
6. Baris 54 mengembalikan nilai variabel populasi, yang artinya menampilkan kromosom sebanyak populasi.

```
1.  public int randomDenganPengecualian(Random rnd, int
2.  start, int end, ArrayList<Integer> exclude) {
3.
4.  int arr[] = new int[exclude.size()];
5.
6.   for (int i = 0; i < exclude.size(); i++) {
7.    arr[i] = exclude.get(i);
8.    }
9.    int random = start + rnd.nextInt(end - start + 1
```

```
10.     arr.length);
11.
12.    for (int ex : arr) {
13.     if (random < ex) {
14.      break;
15.     }
16.     random++;
17.    }
18.    return random;
19.       }
```

Kode Program 5.2 Implementasi *Random* Dengan Pengecualian

Proses yang dilakukan dalam *random* dengan pengecualian pada Kode Program 5.2 antara lain:

1. Baris 4 untuk inisialisasi awal variabel arr[].
2. Baris 6-8 perulangan untuk mendapatkan nilai apa saja yang termasuk *exclude* dan ditambahkan pada variabel arr[].
3. Baris 10-11 inisialisasi variabel *random*
4. Baris 13-28 untuk mendapatkan nilai *random*.
5. Baris 19 pengembalian nilai pada variabel *random*.

5.1.2 Implementasi *Crossover*

Proses *crossover* dilakukan dengan mengkawin silangkan 2 buah individu yang diambil secara *random* sebagai *parent* dalam menghasilkan keturunan. Metode yang digunakan adalah *one-cut point crossover* yaitu menukarkan susunan kromosom denga satu titik potong yang mana menggabungkan beberapa gen dari induk pertama dan kedua. Implementasi *crossover* dapat dilihat pada Kode Program 5.3.

```
1.  public double[][] crossover(double[][] p) {
2.
3.  double jum_cros = Math.ceil((double) this.anak_cros / 2);
4.  double anak_cros[][] = new
5.  double[this.anak_cros][p[0].length];
6.  Random rand = new Random();
7.  ArrayList temp_anak = new ArrayList();
8.
9.   for (int i = 0; i < jum_cros; i++) {
10.   int p1 = 0 + (int) (Math.random() * (((this.popSize -
11.   1) - 0) + 0));
12.   ArrayList<Integer> pengecualianRandom = new
13.   ArrayList<>();
14.   pengecualianRandom.add(p1);
15.   int p2 = randomDenganPengecualian(rand, 0, this.popSize
16.   - 1, pengecualianRandom);
17.
18.    // random titik potong
19.    int cutPoint[] = new int[this.jumlahKelas];
20.    for (int j = 0; j < cutPoint.length; j++) {
21.     cutPoint[j] = this.batasBawahGen + (int)
22.     (Math.random() * ((batasAtas[j] - this.batasBawahGen)
```

```
23.      + 1));
24.      }
25.      temp_anak.add(transformAndCross(p[p1], p[p2],
26.      cutPoint));
27.      temp_anak.add(transformAndCross(p[p2], p[p1],
28.      cutPoint));
29.     }
30.
31.     for (int i = 0; i < this.anak_cros; i++) {
32.      Object temp = temp_anak.get(i);
33.      anak_cros[i] = (double[]) temp;
34.     }
35.    return anak_cros;
36.   }
37.
38.
```

Kode Program 5.3 Implementasi *Crossover*

Proses yang dilakukan dalam *crossover* pada Kode Program 5.2 antara lain:

1. Baris 3-7 inisialisasi variabel jum_cros, anak_cros[][], rand, dan temp_anak.
2. Baris 18-24 untuk menentukan titik pototng secara *random*.
3. Baris 9-29 untuk menentukan *parent* dalam proses *crossover*.
4. Baris 31-34 untuk menyimpan anak dari hasil *crossover*.
5. Baris 35 mengembalikan nilai dari variabel anak_cros.

Proses yang dilakukan dalam *transformAndCross* pada Kode Program 5.4 antara lain:

1. Baris 4 -5 untuk inisialisasi variabel c[] dan temp_index.
2. Baris 10-15 untuk menentukan titik potong pada p1 dan p2.
3. Baris 17-20 untuk menghapus nilai p2 yang ada pada p1.
4. Baris 22-25 untuk menggabungkan nilai dari p1 dan p2.
5. Baris 27-30 untuk menggabungkan menjadi 1 kromosom lagi.
6. Baris 7-32 untuk membentuk kromosom yang akan di *crossover*.
7. Baris 33 pengembalian nilai variabel c.

5.1.3 Implementasi Mutasi

Proses mutasi hanya menggunakan 1 individu sebagai induk yang dipilih secara *random*. Mutasi dilakukan dengan mengubah susunan gen pada kromosom, hasil perubahan tersebut akan menjadi keturunannya. Jumlah keturunan yang dihasilkan, ditentukan dari nilai *mr* (*mutation rate*) yang dibentuk antara 0 sampai 1. Implementasi mutasi dapat dilihat pada Kode Program 5.5.

Proses yang dilakukan dalam *Mutation* pada Kode Program 5.3 antara lain:

1. Baris 2-4 untuk inisialisasi variabel anak_mutasi dan rand.

2. Baris 11-19 untuk menemtukan titik tukar yang akan dimasukkan ke dalam pengecualian*Random* dan memanggil method *random*DenganPengecualian.
3. Baris 6-22 untuk mendapatkan *child* dari proses mutasi.
4. Baris 23 penngembalian nilai variabel anak_mutasi.

```
1.  Public double[] transformAndCross(double p1[], double
2.  p2[], int cutPoint[]) {
3.
4.  double c[] = new double[this.jumlahKromosom + 1];
5.  int temp_index = 0;
6.
7.   for (int i = 0; i < this.jumlahKelas; i++) {
8.    ArrayList temp_perkelas_p1 = new ArrayList();
9.    ArrayList temp_perkelas_p2 = new ArrayList();
10.   for (int j = 0; j < this.panjangGenKelas[i]; j++) {
11.    if (j < cutPoint[i]) {
12.     temp_perkelas_p1.add(p1[j + temp_index]);
13.    }
14.    temp_perkelas_p2.add(p2[j + temp_index]);
15.   }
16.
17.   //hapus p2 yang ada di p1
18.   for (int j = 0; j < temp_perkelas_p1.size(); j++) {
19.    temp_perkelas_p2.remove(temp_perkelas_p1.get(j));
20.   }
21.
22.   //gabung p1 dan p2
23.   ArrayList temp_c_perkelas = new
24.   ArrayList(temp_perkelas_p1);
25.   temp_c_perkelas.addAll(temp_perkelas_p2);
26.
27.   //gabung jadi 1 kromosom lagi
28.   for (int j = 0; j < temp_c_perkelas.size(); j++) {
29.    c[j + temp_index] = (double) temp_c_perkelas.get(j);
30.   }
31.   temp_index += this.panjangGenKelas[i];
32.  }
33.  return c;
34. }
```

Kode Program 5.4 Implementasi *transformAndCross*

```
1.  public double[][] mutasi(double p[][]) {
2.  double[][] anak_mutasi = new
3.  double[this.anak_mut][p[0].length];
4.  Random rand = new Random();
5.
6.   for (int i = 0; i < this.anak_mut; i++) {
7.    int p1 = 0 + (int) (Math.random() * ((this.popSize -
8.    0)));
9.    int titikTukar[][] = new int[this.jumlahKelas][2];
10.
11.   for (int j = 0; j < titikTukar.length; j++) {
12.    titikTukar[j][0] = this.batasBawahGen + (int)
13.    (Math.random() * (((batasAtas[j] - 1) - 0) + 0));
14.    ArrayList<Integer> pengecualianRandom = new
15.    ArrayList<>();
```

```
16.     pengecualianRandom.add(titikTukar[j][0]);
17.     titikTukar[j][1] = randomDenganPengecualian(rand, 0,
18.     this.batasAtas[j] - 1, pengecualianRandom);
19.    }
20.
21.    anak_mutasi[i] = transformAndMutate(p[p1], titikTukar);
22.   }
23.   return anak_mutasi;
24.  }
```

Kode Program 5.5 Implementasi Mutasi

```
1.   public double[] transformAndMutate(double p1[], int
2.   titikTukar[][]) {
3.
4.   double c[] = new double[this.jumlahKromosom + 1];
5.
6.   // mengubah bentuk array p1
7.   int temp_index = 0;
8.   for (int i = 0; i < this.jumlahKelas; i++) {
9.    double[] temp_c = new double[this.panjangGenKelas[i]];
10.
11.    for (int j = 0; j < temp_c.length; j++) {
12.     temp_c[j] = p1[j + temp_index];
13.    }
14.
15.    swap(temp_c, titikTukar[i][0], titikTukar[i][1]);
16.
17.    for (int j = 0; j < this.panjangGenKelas[i]; j++) {
18.     c[j + temp_index] = temp_c[j];
19.    }
20.
21.    temp_index += this.panjangGenKelas[i];
22.   }
23.   return c;
24.  }
```

Kode Program 5.6 Implementasi *transformAndMutate*

Proses yang dilakukan dalam *transformAndMutate* pada Kode Program 5.6 antara lain:

1. Baris 4 untuk inisialisasi variabel c[].
2. Baris 6-22 mengubah bentuk p1 untuk menentukan titik potong mutasi dan memanggil method *swap* sehingga didapatkan *child* hasil mutasi.
3. Baris 23 pengembalian nilai variabel c.

Proses yang dilakukan dalam *swap* pada Kode Program 576 antara lain:

1. Baris 1-3 merupakan proses *swap* untuk titik potong yang telah ditentukan pada proses mutasi.

```
1.  public void swap(double[] arr, int i, int j) {
2.    arr[i] = (arr[i] + arr[j]) - (arr[j] = arr[i]);
3.  }
```

Kode Program 5.7 Implementasi *Swap*

5.1.4 Implementasi Perhitungan *Constraint*

Proses perhitungan *constraint* dilakukan untuk megetahui seberapa banyak pelanggaran yang dilanggar pada setiap individu. Setiap pelanggaran yang dilakukan memiliki bobot untuk dihitung nilai *fitness*nya pada proses evaluasi. Implementasi perhitungan *constraint* dapat dilihat pada Kode Program 5.8 – 5.14.

```
1.  public double hitungConstraint1(int ind) {
2.  double f1 = 0;
3.  int counter = 0;
4.
5.   for (int i = 0; i < jumlahJam; i++) {
6.    int ar[] = new int[jumlahKelas];
7.    int temp_index = 0;
8.
9.     for (int j = 0; j < jumlahKelas; j++) {
10.     ar[j] = (int) this.pKTM[ind][i + temp_index];
11.     temp_index += jumlahJam;
12.    }
13.
14.    if (cekBentrok(ar) == 1) {
15.     f1 += this.bobotConstraint1;
16.     counter++;
17.    }
18.
19.   }
20.
21.   this.constrain[0] = counter;
22.   return f1;
23.  }
```

Kode Program 5.8 Implementasi Hitung *Constraint* Ke-1

Proses yang dilakukan dalam hitung *constraint* ke-1 pada Kode Program 5.8 antara lain:

1. Baris 2-3 untuk inisialisasi variabel f1 dan *counter*.
2. Baris 9-12 untuk mendapatkan jumlah jam berdasarkan kode tugas mengajar.
3. Baris 14-17 untuk mengecek apakah ada bentrok pada tugas mengajar. Jika ada maka akan menambahkan bobot *constraint* pada variabel f1 dan jumlah pelanggaran pada *counter* bertambah.
4. Baris 5-19 untuk mendapatkan kode tugas mengajar sebanyak jumlah jam.
5. Baris 21-22 mengembalikan nilai *counter* dan f1.

```
1.  public int cekBentrok(int ar[]) {
2.   int status = 0;
3.
4.    for (int i = 0; i < ar.length; i++) {
5.     for (int j = i + 1; j < ar.length; j++) {
6.      if (ar[i] == ar[j] && (ar[i] != -1 || ar[j] != -1)) {
7.       status = 1;
8.       break;
9.      }
10.    }
11.   }
12.
13.   return status;
14.  }
```

Kode Program 5.9 Implementasi cekBentrok

Proses yang dilakukan dalam cek bentrok pada Kode Program 5.9 antara lain:

1. Baris 2 untuk inisialisasi awal variabel status.
2. Baris 4—11 untuk melihat apakah terjadi bentrok, jika iya maka nilai pada variabel status berubah menjadi 1.
3. Baris 13 pengembalian nilai variabel status.

```
1.  public double hitungConstraint2(int ind) {
2.  double f2 = 0;
3.  int counter = 0;
4.  int indeksGen = 0;
5.
6.   for (int i = 0; i < this.jumlahKelas; i++) {
7.    for (int j = 0; j < this.jumlah_jam_perhari.length;
8.    j++) {
9.     for (int k = 0; k < this.jumlah_jam_perhari[j]; k++) {
10.     if (j == 0) {
11.      if (this.pKMP[ind][indeksGen++] == 0 && k + 2 > 4) {
12.       f2 += this.bobotConstraint2;
13.       counter++;
14.      }
15.     } else {
16.      if (this.pKMP[ind][indeksGen++] == 0 && k + 1 > 4) {
17.       f2 += this.bobotConstraint2;
18.       counter++;
19.      }
20.     }
21.    }
22.   }
23.  }
24.
25.  this.constrain[1] = counter;
26.  return f2;
27. }
```

Kode Program 5.10 Implementasi Hitung *Constraint* Ke-2

Proses yang dilakukan dalam hitung *Constraint* ke-2 pada Kode Program 5.10 antara lain:

1. Baris 2-4 untuk inisialisasi variabel f2, *counter*, dan indeksGen.
2. Baris 9-21 untuk mengetahui apakah kode mata pelajaran penjasker OR berada di atas jam ke-4. Jika iya maka menambahkan bobot *constraint* pada variabel f2 dan jumlah pelanggaran pada *counter* bertambah.
3. Bairs 7-22 untuk mengecek disetiap harinya.
4. Baris 6-23 untuk mengecek disetiap kelasnya.
5. Baris 25-26 mengembalikan nilai *counter* dan f2.

```
1.  public double hitungConstraint3(int ind) {
2.  double f3 = 0;
3.  int counter = 0;
4.  int indeksGen = 0;
5.  int inisialIndeks = 0;
6.
7.   for (int i = 0; i < this.jumlahKelas; i++) {
8.    for (int j = 0; j < this.jumlah_jam_perhari.length;
9.    j++) {
10.    int panjang = 0;
11.     for (int k = 0; k < jumlah_jam_perhari[j]; k++) {
12.      panjang++;
13.      indeksGen++;
14.     }
15.     double hari[] = new double[panjang];
16.     System.arraycopy(this.pKMP[ind], inisialIndeks, hari,
17.     0, panjang);
18.     inisialIndeks += panjang;
19.     int lebih4[] = cekJamMengajarLebih4Jam(hari);
20.     if (lebih4[0] == 1) {
21.      if (lebih4[1] == 0) {
22.       f3 += this.bobotConstraint3;
23.       counter++;
24.      } else {
25.        f3 += (((double) 1 / (1 + lebih4[1])) *
26.        this.bobotConstraint3);
27.        counter++;
28.       }
29.      }
30.    }
31.   }
32.
33.   this.constrain[2] = counter;
34.   return f3;
35.  }
```

Kode Program 5.11 Implementasi Hitung *Constraint* Ke-3

Proses yang dilakukan dalam hitung c*onstraint* ke-3 pada Kode Program 5.11 antara lain:

1. Baris 2-5 untuk inisialisasi awal variabel f3, *counter*, indeksgen, dan inisialIndeks.

2. Baris 8-30 untuk mengecek apakah terdapat mata pelajaran yang berjumlah lebih dari 4 jam dalam satu hari. Jika iya maka menambahkan bobot *constraint* pada variabel f3 dan jumlah pelanggaran pada *counter* bertambah.
3. Baris 7-31 untuk mengecek apakah terdapat mata pelajaran yang berjumlah lebih dari 4 jam pada setiap kelas.
4. Baris 33-34 mengembalikan nilai *counter* dan f3.

```
1.  public static int[] cekJamMengajarLebih4Jam(double
2.  hari[]) {
3.
4.  int status = 0;
5.  int nilaiSama = 0;
6.  int hasil[] = new int[2];
7.  int jeda = 0;
8.  int indeksTerakhir = 0;
9.
10. for (int i = 0; i < hari.length; i++) {
11.  for (int j = i + 1; j < hari.length; j++) {
12.   if (hari[i] == hari[j] && hari[i] != -1) {
13.     nilaiSama++;
14.     indeksTerakhir = j;
15.    } else if (hari[i] != hari[j] && hari[i] != -1) {
16.     jeda++;
17.   }
18.  }
19.  if (nilaiSama > 3) {
20.   status = 1;
21.   break;
22.  } else {
23.     nilaiSama = 0;
24.     jeda = 0;
25.   }
26.  }
27.
28.  hasil[0] = status;
29.  hasil[1] = jeda - (hari.length - (indeksTerakhir + 1));
30.  return hasil;
31. }
```

Kode Program 5.12 Implementasi cekJamMengajarLebih4 Jam

Proses yang dilakukan dalam cek jam mengajar lebih 4 jam pada Kode Program 5.12 antara lain:

1. Baris 4-8 untuk inisialisasi variabel status, nilaiSama, hasil[], jeda, dan indeksTerakhir.
2. Baris 11-18 untuk mengecek apakah ada nilai yang sama secara sekuensial dan terdpat jeda atau tidak.
3. Baris 10-26 apabila terdapat nilai yang sama lebih dari 3 maka status berubah menjadi 1, jika tidak maka status tetap 0.
4. Baris 28-229 menampilkan jumlah status dan jeda, serta mengembalikan nilai dari variabel hasil.

```
1.  public double hitungConstraint4(int ind) {
2.  double f4 = 0;
3.  int counter = 0;
4.  int jumlahhari = 2;
5.  int temp_index = 0;
6.
7.  for (int i = 0; i < this.jumlahKelas; i++) {
8.   for (int j = 0; j<this.jumlah_jam_perhari.length;j++) {
9.    temp_index += jumlah_jam_perhari[j];
10.   if (j < jumlahhari - 1) {
11.    int hari_ini_jam_terakhir = (int)
12.    this.pKMP[ind][temp_index - 1];
13.    int besok_jam_pertama = (int)
14.    this.pKMP[ind][temp_index];
15.     if (hari_ini_jam_terakhir == besok_jam_pertama) {
16.      f4 += this.bobotConstraint4;
17.      counter++;
18.     }
19.    }
20.   }
21.  }
22.
23.  this.constrain[3] = counter;
24.  return f4;
25. }
```

Kode Program 5.13 Implementasi Hitung *Constraint* Ke-4

Proses yang dilakukan dalam hitung *constraint* ke-4 pada Kode Program 5.13 antara lain:

1. Baris 2-5 inisialisasi variabel f4, *counter*, jumlahhari, dan temp_index.
2. Baris 8-20 untuk mengecek apakah ada mata pelajaran yang sama pada jam kedua dan selanjutnya (sejumlah ketentuan) berada pada hari yang berbeda. Jika iya maka menambahkan bobot *constraint* pada variabel f4 dan jumlah pelanggaran pada *counter* bertambah.
3. Baris 7-21 untuk mengecek apakah ada mata pelajaran yang sama pada jam kedua dan selanjutnya (sejumlah ketentuan) berada pada hari yang berbeda pada setiap kelas.
4. Bairs 23-24 mengembalikan nilai *counter* dan f4.

Proses yang dilakukan dalam hitung *fitness* pada Kode Program 5.14 antara lain:

1. Baris 2 untuk inisialisasi variabel *fitness*.
2. Baris 4-7 untuk menghitung bobot pada setiap pelanggaran.
3. Baris 9 untuk penjumlahan seluruh bobot pelanggaran..
4. Baris 10 untuk menghitung nilai *fitness* berdasarkan jumlah bobot pelanggaran.
5. Baris 12 mengembalikan nilai variabel *fitness*.

```
1.  public double hitungFitness(int individu) {
2.   double fitness = 0, f1, f2, f3, f4, fx;
3.
4.   f1 = hitungConstraint1(individu);
5.   f2 = hitungConstraint2(individu);
6.   f3 = hitungConstraint3(individu);
7.   f4 = hitungConstraint4(individu);
8.
9.   fx = f1 + f2 + f3 + f4;
10.  fitness = 1 / (1 + fx);
11.
12.  return fitness;
13. }
```

Kode Program 5.14 Implementasi Hitung *Fitness*

5.1.5 Implementasi Evaluasi

Pada proses evaluasi dilakukan penggabungan seluruh individu dalam populasi beserta keturunannya dengan nilai *fitness* yang dimiliki pada setiap individu. Selanjutnya akan dilakukan evaluasi nilai *fitness* pada setiap individu, individu dengan nilai tertinggi memiliki kemungkinan besar untuk dapat masuk pada generasi berikutnya. Implementasi evaluasi dapat dilihat pada Kode Program 5.15.

```
1.  public double[][] evaluasi(double[][] p, double[][]
2.  anak_cros, double[][] anak_mut) {
3.
4.  double[][] populasi_akhir = new double[p.length +
5.  anak_cros.length + anak_mut.length][];
6.  System.arraycopy(p, 0, populasi_akhir, 0, p.length);
7.  System.arraycopy(anak_cros, 0, populasi_akhir, p.length,
8.  anak_cros.length);
9.
10. for (int i = 0; i < anak_mut.length; i++) {
11.  populasi_akhir[i + p.length + anak_cros.length] =
12. anak_mut[i];
13. }
14.
15. ekstraksiKromosom(populasi_akhir);
16.
17. for (int i = 0; i < populasi_akhir.length; i++) {
18.  populasi_akhir[i][populasi_akhir[i].length - 1] =
19.  hitungFitness(i);
20. }
21.
22.  return populasi_akhir;
23. }
```

Kode Program 5.15 Implementasi Evaluasi

Proses yang dilakukan dalam evaluasi pada Kode Program 5.8 antara lain:

1. Baris 4-8 untuk inisialisasi populasi_akhir dan meng-*copy* array.
2. Baris 10-13 untuk menampilkan jumlah *child* dari hasil mutasi.
3. Baris 15 memanggil method ekstraksi kromosom.

4. Bairs 17-20 untuk menampilkan keseluruhan populasi dan menghitung nilai *fitness* pada masing-masing individu didalamnya.
5. Baris 22 pengembalian nilai variabel populasi_akhir.

```
1.  public void ekstraksiKromosom(double ek[][]) {
2.  this.pKTM = new double[ek.length][jumlahJam *
3.  jumlahKelas];
4.  this.pKMP = new double[ek.length][jumlahJam *
5.  jumlahKelas];
6.
7.  for (int i = 0; i < ek.length; i++) {
8.   int temp_index = 0;
9.   int temp_index1 = 0;
10.
11.  for (int j = 0; j < this.jumlahKelas; j++) {
12.   double temp_kromosom_perkelas[] = new
13.   double[this.panjangGenKelas[j]];
14.
15.   for (int k = 0; k < this.panjangGenKelas[j]; k++) {
16.    temp_kromosom_perkelas[k] = ek[i][k + temp_index];
17.   }
18.   double temp_ktm_perkelas[] =
19.   get_KTM_or_KMP(temp_kromosom_perkelas, j, "ktm");
20.   double temp_kmp_perkelas[] =
21.   get_KTM_or_KMP(temp_kromosom_perkelas, j, "kmp");
22.
23.   for (int k = 0; k < temp_ktm_perkelas.length; k++) {
24.    this.pKTM[i][k + temp_index1] = temp_ktm_perkelas[k];
25.    this.pKMP[i][k + temp_index1] = temp_kmp_perkelas[k];
26.   }
27.
28.   temp_index1 += temp_ktm_perkelas.length;
29.   temp_index += this.panjangGenKelas[j];
30.   }
31.  }
32. }
```

Kode Program 5.16 Implementasi ekstraksiKromosom

Proses yang dilakukan dalam ekstraksi kromosom pada Kode Program 5.16 antara lain:

1. Baris 2-5 untuk inisialisasi variabel KTM dan KMP.
2. Baris 15-17 untuk mendapatkan panjang kromosom perkelas.
3. Baris 23-26 untuk mendapatkan kode tugas mengajar dan kode mata pelajaran perkelas.
4. Baris 11-30 untuk mendapatkan kode tugas mengajar dan kode mata pelajaran sejumlah kelas.
5. Baris 7-31 untuk mendapatkan kode tugas mengajar dan kode mata pelajaran sepanjang kromosom.

Proses yang dilakukan dalam *get_KTM_or_KMP* pada Kode Program 5.17 antara lain:

1. Baris 4-6 untuk inisialisasi variabel temp_index, *counter*, dan all_day.
2. Baris 8-10 untuk set variabel all_day index ke *i* sebesar -99.
3. Baris 12-31 untuk mengisi kode tugas mengajar atau ode mata pelajaran pada setiap kromosom perkelas dengan memanggil method ambilKodeTugasMengajar dan ambilKodeMataPelajaran.
4. Baris 32 pengembalian nilai variabel all_day.

```
1.  public double[] get_KTM_or_KMP(double[]
2.  temp_kromosom_perkelas, int kelas, String kode) {
3.
4.  int temp_index = 0;
5.  int counter = 0;
6.  double[] all_day = new double[jumlahJam];
7.
8.  for (int i = 0; i < all_day.length; i++) {
9.   all_day[i] = -99;
10. }
11.
12. for (int i = 0; i < temp_kromosom_perkelas.length; i++) {
13.  if (temp_kromosom_perkelas[i] == 0) {
14.   all_day[counter] = -1;
15.   counter++;
16.  } else {
17.   int temp_kode_jumlahJam[] = null;
18.   if (kode.equalsIgnoreCase("ktm")) {
19.    temp_kode_jumlahJam = ambilKodeTugasMengajar(kelas,
20.    (int) temp_kromosom_perkelas[i]);
21.   } else {
22.    temp_kode_jumlahJam = ambilKodeMataPelajaran(kelas,
23.    (int) temp_kromosom_perkelas[i]);
24.    }
25.    for (int j = 0; j < temp_kode_jumlahJam[1]; j++) {
26.     all_day[counter] = temp_kode_jumlahJam[0];
27.     counter++;
28.    }
29.   }
30.
31.  }
32.  return all_day;
33. }
```

Kode Program 5.17 Implementasi *get_KTM_or_KMP*

```
1.  public int[] ambilKodeTugasMengajar(int kelas, int
2.  kodeKelas) {
3.
4.  int kodeTM = 0;
5.  int counter = 0;
6.  int jumlah_jam = 0;
7.  int index_kelas;
8.
9.  for (int i = 0; i < this.dataTugasMengajar.length; i++) {
10.  if (this.dataTugasMengajar[i][2] == kelas) {
11.   counter++;
```

```
12.    }
13.    if (counter == kodeKelas) {
14.     kodeTM = i;
15.     jumlah_jam = this.dataTugasMengajar[i][3];
16.     break;
17.    }
18.   }
19.
20.    int kode_jumlahJam[] = {kodeTM, jumlah_jam};
21.    return kode_jumlahJam;
22.   }
```

Kode Program 5.18 Implementasi Ambil Kode Tugas Mengajar

Proses yang dilakukan dalam ambil kode tugas mengajar pada Kode Program 5.18 antara lain:

1. Baris 4-7 inisialisasi variabel kodeTM, *counter*, jumlah_jam, dan index_kelas.
2. Baris 9-18 untuk mendapatkan kode tugas mengajar beserta jumlah jamnya.
3. Baris 20 untuk memasukkan kodeTM ke dalam jumlah_jam pada kode_jumlahJam[].
4. Baris 21 pengembalian nilai variabel kode_jumlahJam.

```
1.   public int[] ambilKodeMataPelajaran(int kelas, int
2.   kodeKelas) {
3.
4.   int kodeTM = 0;
5.   int counter = 0;
6.   int jumlah_jam = 0;
7.
8.   for (int i = 0; i < this.dataTugasMengajar.length; i++) {
9.    if (this.dataTugasMengajar[i][2] == kelas) {
10.    counter++;
11.   }
12.   if (counter == kodeKelas) {
13.    kodeTM = (int) this.dataTugasMengajar[i][1];
14.    jumlah_jam = this.dataTugasMengajar[i][3];
15.    break;
16.   }
17.  }
18.
19.   int kode_jumlahJam[] = {kodeTM, jumlah_jam};
20.   return kode_jumlahJam;
21.  }
```

Kode Program 5.19 Implementasi Ambil Kode Mata Pelajaran

Proses yang dilakukan dalam ambil kode mata pelajaran pada Kode Program 5.19 antara lain:

1. Baris 4-6 inisialisasi variabel kodeTM, *counter*, dan jumlah_jam.
2. Baris 8-17 untuk mendapatkan kode mata pelajaran beserta jumlah jamnya.
3. Baris 19 untuk memasukkan kodeTM ke dalam jumlah_jam pada kode_jumlahJam[].
4. Baris 20 pengembalian nilai variabel kode_jumlahJam.

5.1.6 Implementasi Seleksi

Proses seleksi pada penelitian ini meggunakan metode *elitism selection*, yang mana akan dilakukan pemilihan terhadap individu yang mempunyai nilai *fitness* tertinggi sejumlah populasi awal berdasarkan hasil evaluasi sebelumnya. Individu hasil seleksi ini akan gunakan sebagai individu pada populasi di generasi selanjutnya. Implementasi seleksi dapat dilihat pada Kode Program 5.20.

```
1.  public double[][] seleksi(double[][] populasi) {
2.  double tempSeleksi[][] = new
3.  double[populasi.length][populasi[0].length];
4.  double hasilSeleksi[][] = new
5.  double[this.popSize][populasi[0].length];
6.
7.  for (int i = 0; i < populasi.length; i++) {
8.   System.arraycopy(populasi[i], 0, tempSeleksi[i], 0,
9.   populasi[0].length);
10. }
11.
12.  tempSeleksi = elitism(tempSeleksi);
13.  System.arraycopy(tempSeleksi, 0, hasilSeleksi, 0,
14.  this.popSize);
15.  return hasilSeleksi;
16. }
```

Kode Program 5.20 Implementasi Seleksi

Proses yang dilakukan dalam seleksi pada Kode Program 5.20 antara lain:

1. Baris 2-5 inisialisasi variabel tempSeleksi[][] dan hasilSeleksi[][].
2. Baris 7-10 untuk meng-*copy* array sebanyak jumlah populasi.
3. Baris 12 melakukan seleksi dengan memanggil method *elitism*.
4. Baris 13 meng-*copy* array hasil seleksi.
5. Baris 14 mengembalikan nilai variabel hasilSeleksi.

```
1.  public double[][] elitism(double[][] pSeleksi) {
2.  double[][] tempP = pSeleksi;
3.
4.  Arrays.sort(tempP, new java.util.Comparator<double[]>() {
5.   public int compare(double[] a, double[] b) {
6.    return Double.compare(b[b.length - 1], a[a.length -
7.    1]);
8.    }
9.   });
10.
11.  return tempP;
12. }
```

Kode Program 5.21 Implementasi *Elitism*

Proses yang dilakukan dalam *elitism* pada Kode Program 5.21 antara lain:

1. Baris 2 inisialisasi variabel temP.

2. Baris 4-9 menyorting variabel tempP dengan membandingkan antara 2 individu, a dan b.

3. Baris 11 pengembalian nilai variabel tempP.

5.1.7 Implementasi *Simulated Annealing*

Individu terbaik yang dihasilkan dari algoritme genetika selanjutnya akan dilakukan proses perhitungan dengan menggunakan *simulated annealing.* Beberapa proses utama dari *simulated annealing* antara lain *neighborhood move*, evaluasi, dan *probability acceptance*. Implementasi *simulated annealing* dapat dilihat pada Kode Program 5.22.

```
1.  double[] SA(double[] individu_GA) {
2.  double individu[] = new double[individu_GA.length];
3.  System.arraycopy(individu_GA, 0, individu, 0,
4.  individu.length);
5.  double taksen = this.t_awal;
6.  int counter = 0;
7.  int counter1 = 0;
8.
9.  while (taksen <= t_akhir) {
10.  if (counter == 4 || counter1 == 4) {
11.   break;
12.  }
13.  double temp_ind[] = neigborhoodMove(individu);
14.  double delta_fitnes = temp_ind[temp_ind.length - 1] -
15.  individu[individu.length - 1];
16.  if (delta_fitnes < 0) {
17.   individu = temp_ind;
18.   counter++;
19.  } else {
20.    double toleransi = 1 / 1 + Math.exp(-delta_fitnes /
21.    this.t_awal);
22.     if (toleransi > Math.random()) {
23.      individu = temp_ind;
24.      counter1++;
25.     } else {
26.       taksen *= this.beta;
27.     }
28.    }
29.   }
30.
31.  return individu;
32. }
```

Kode Program 5.22 Implementasi *Simulated Annealing*

Proses yang dilakukan dalam *simulated annealing* pada Kode Program 5.22 antara lain:

1. Baris 2-7 inisialisasi variabel individu[], taksen, *counter*, *counter*1, dan meng-*copy* array.

2. Baris 9-29 kondisi di mana selama taksen ≤ takhir maka akan dilakukan proses neighborhood dan probability acceptance untuk perbaikan individu setalah itu dilakukan penurunan suhu.

3. Baris 31 pengembalian nilai variabel individu.

```
1.  public double[] neigborhoodMove(double[] nm) {
2.  Random rand = new Random();
3.  double tempNM[] = new double[nm.length];
4.  System.arraycopy(nm, 0, tempNM, 0, nm.length);
5.  int titikTukar[][] = new int[this.jumlahKelas][2];
6.
7.  for (int i = 0; i < titikTukar.length; i++) {
8.   titikTukar[i][0] = this.batasBawahGen + (int)
9.   (Math.random() * (((batasAtas[i] - 1) - 0) + 0));
10.  ArrayList<Integer> pengecualianRandom = new
11.  ArrayList<>();
12.  pengecualianRandom.add(titikTukar[i][0]);
13.  titikTukar[i][1] = randomDenganPengecualian(rand, 0,
14.  this.batasAtas[i] - 1, pengecualianRandom);
15. }
16.
17.  tempNM = transformAndMutate(tempNM, titikTukar);
18.  tempNM = evaluasiSA(tempNM);
19.  return tempNM;
20. }
```

Kode Program 5.23 Implementasi *Neighborhood Move*

Proses yang dilakukan dalam *neighborhood move* pada Kode Program 5.23 antara lain:

1. Baris 2-5 inisialisasi variabel rand, tempNM[], titikTukar[][], dan meng-*copy* array.
2. Baris 7-15 untuk menentukan titik tukar pada individu terbaik hasil algoritme genetika
3. Baris 17 untuk memanggil method transformAndMutate untuk mutasi individu dengan titik tukar yang telah ditentukan sebelumnya.
4. Baris 18 untuk menghitung nilai *fitness* dengan memanggil method evaluasiSA.
5. Baris 19 pengembalian nilai variabel tempNM

Proses yang dilakukan dalam evaluasi SA pada Kode Program 5.24 antara lain:

1. Baris 2-3 inisialisasi variabel tempNM[][] dan mengcoy array.
2. Baris 5 memanggil ekstraksiKromosom.
3. Baris 7-10 untuk menghitung nilai *fitness* pada individu hasil *neighborhood move*.
4. Baris 12 pengembalian nilai variabel tempNM[0].

```
1.  public double[] evaluasiSA(double[] nm) {
2.  double tempNM[][] = new double[1][nm.length];
3.  System.arraycopy(nm, 0, tempNM[0], 0, nm.length);
4.
5.  ekstraksiKromosom(tempNM);
```

```
6.
7.    // hitung fitness
8.    for (int i = 0; i < tempNM.length; i++) {
9.     tempNM[i][tempNM[i].length - 1] = hitungFitness(i);
10.   }
11.
12.    return tempNM[0];
13.   }
```

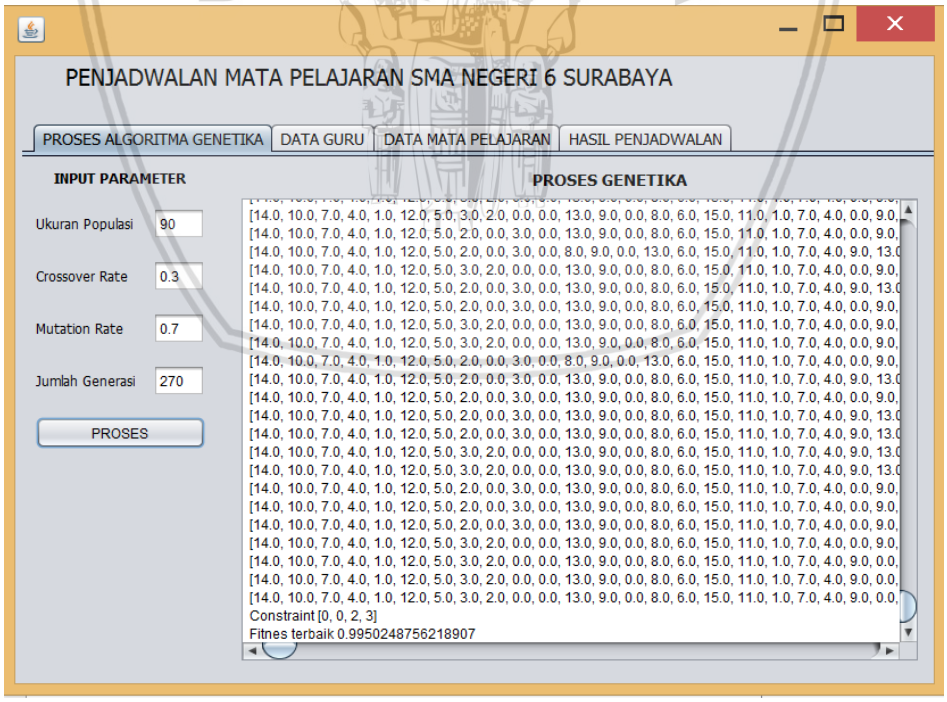
Kode Program 5.24 Implementasi Evaluasi SA

5.2 Implemetasi Antarmuka

Implementasi ini merupakan hasil dari perancangan antarmuka ada bab sebelumnya. Implementasi antarmuka untuk penjadwalan mata pelajaran di SMA Negeri 6 Surabaya ini terdiri dari 4 halaman, yaitu halaman proses hibridisasi *Algoritme* GA-SA, halaman data guru, halaman mata pelajaran, dan halaman hasil penjadwalan.

5.2.1 Implementasi Halaman Proses Hibridisasi Algoritma GA-SA

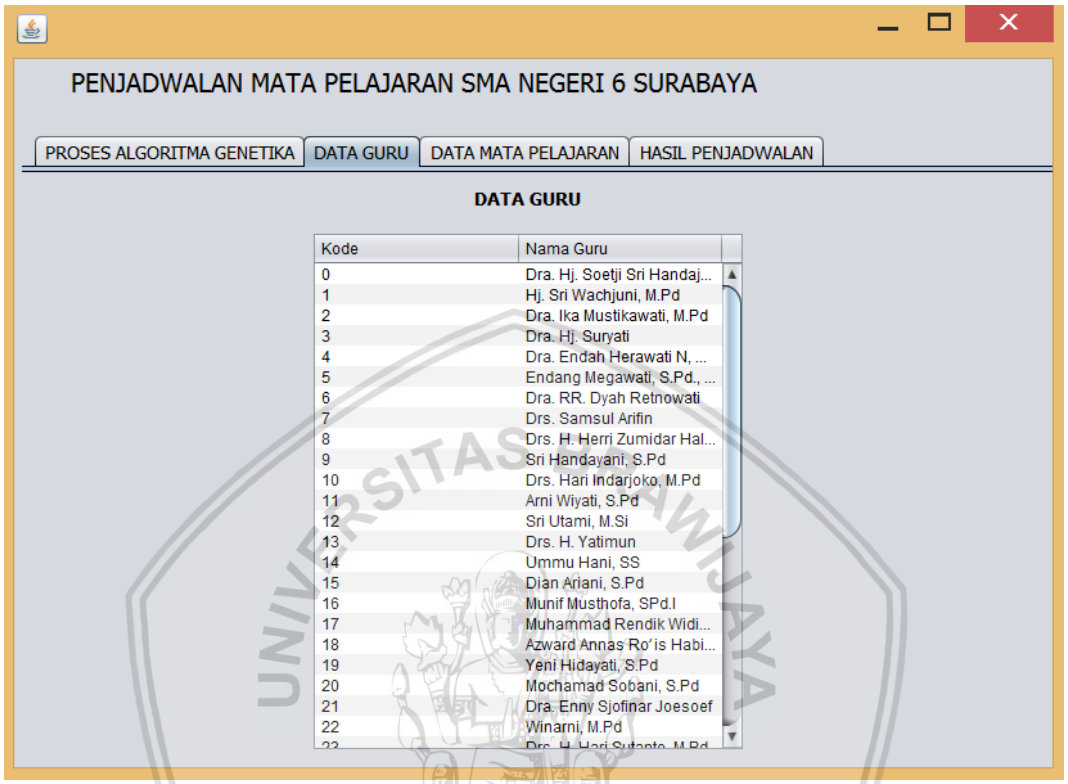
Implementasi pada halaman ini bertujuan mengetahui bagaimana proses hibridisasi *Algoritme* GA-SA dengan memasukan nilai disetiap parameternya seperti jumlah populasi, nilai *cr*, nilai *mr*, dan jumlah generasi. Selanjutnya menekan tombol proses untuk menampilkan proses hibridisasi *Algoritme* GA-SA. Tampilan tersebut dapat dilihat pada Gambar 5.1.



Gambar 5.1 Implementasi Halaman Proses Hibridisasi Algoritma GA-SA

5.2.2 Implementasi Halaman Data Guru

Implementasi pada halaman ini bertujuan mengetahui daftar guru siapa saja yang bertugas mengajar di SMA Negeri 6 Surabaya beserta kodenya. Tampilan tersebut dapat dilihat pada Gambar 5.2.



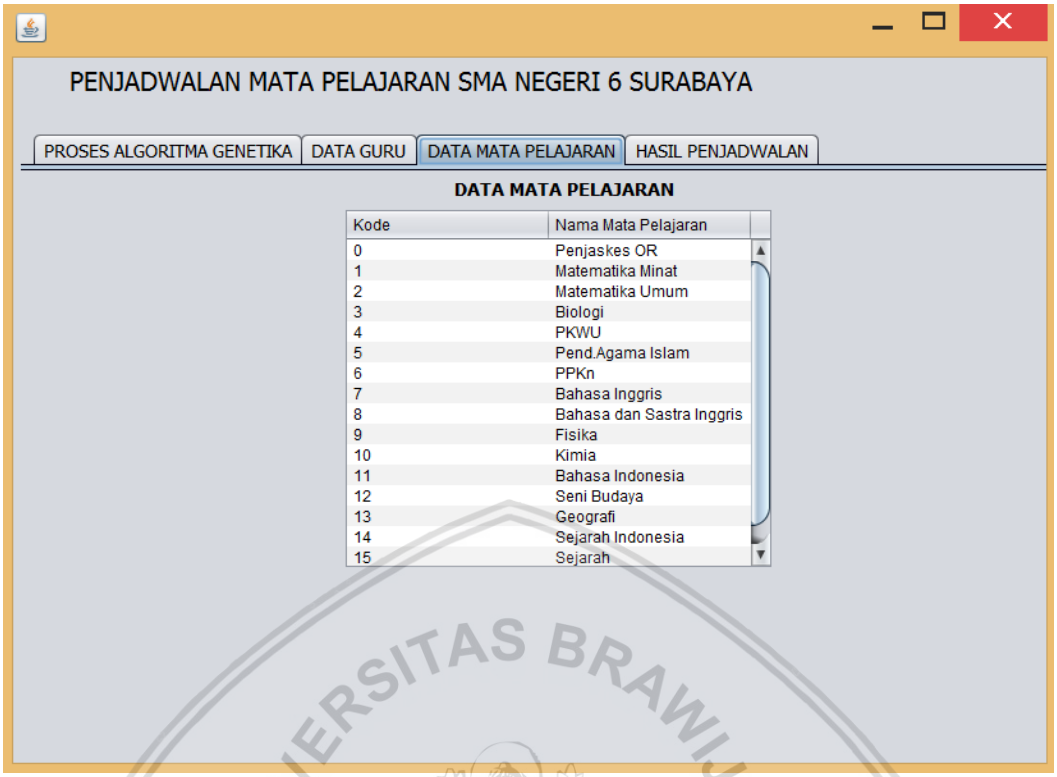
Gambar 5.2 Implementasi Halaman Data Guru

5.2.3 Implementasi Halaman Mata Pelajaran

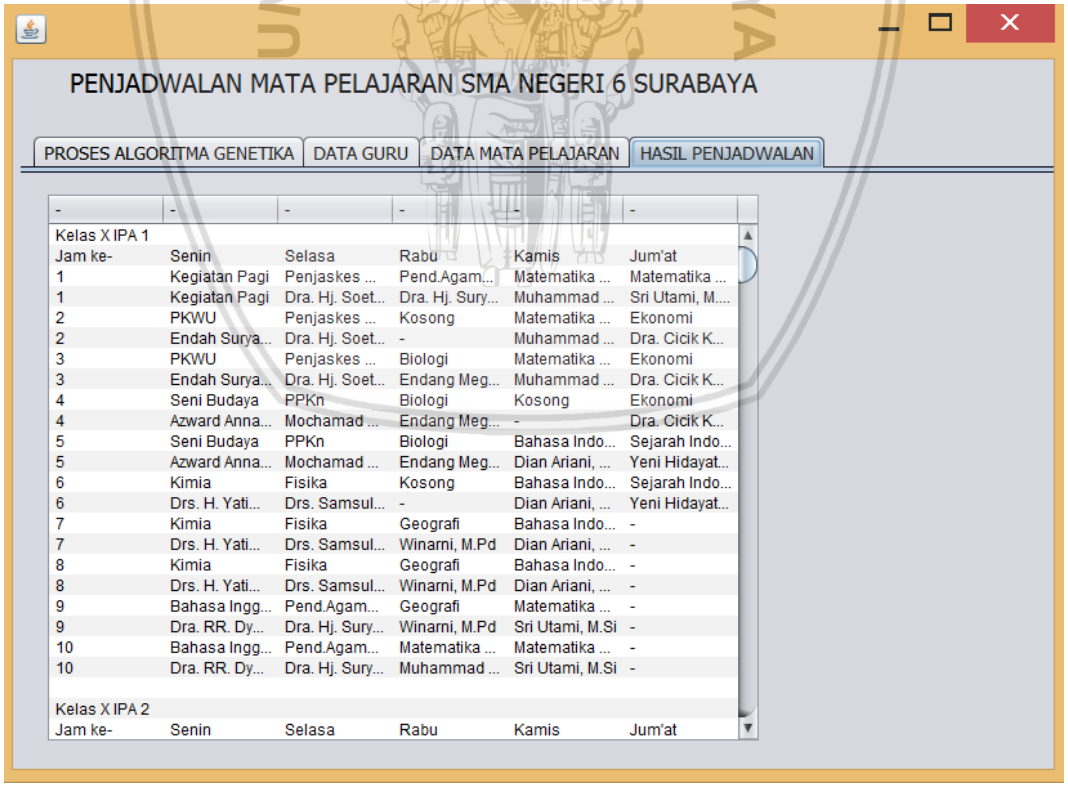
Implementasi pada halaman ini bertujuan mengetahui daftar mata pelajaran apa saja yang ada pada penjadwalan di SMA Negeri 6 Surabaya beserta kodenya. Tampilan tersebut dapat dilihat pada Gambar 5.3.

5.2.4 Implementasi Halaman Hasil Penjadwalan

Implementasi pada halaman ini bertujuan mengetahui hasil penjadwalan dari proses hibridisasi *Algoritme* GA-SA dengan parameter yang telah dimasukan sebelumnya. Tampilan tersebut dapat dilihat pada Gambar 5.4.



Gambar 5.3 Implementasi Halaman Mata Pelajaran



Gambar 5.4 Implementasi Halaman Hasil Penjadwalan

BAB 6 PENGUJIAN DAN ANALISIS

Pada bagian ini akan dibahas tekait proses pengujian terhadap sistem "Optimasi Penjadwalan Mata Pelajaran Pada Kurikulum 2013 dengan Menggunakan Hibridisasi Algoritme Genetika dan *Simulated Annealing* (Studi Kasus: SMA Negeri 6 Surabaya)" yang telah diimplementasikan. Pada proses pengujian dilakukan berdasarkan pada perancangan pengujian yang telah dirancang sebelumnya. Pengujian sistem meliputi pengujian pengaruh generasi, *popSize*, parameter reproduksi, *Algoritme*, dan konvergensi. Dalam semua kondisi setiap pengujian dilakukan sebanyak sepuluh kali running sistem, kemudian diambil nilai rata-rata evaluasinya. Dilakukannya proses pengujian untuk mengetahui hasil evaluasi sistem terhadap implementasi metode yang telah digunakan. Di samping itu juga akan dilakukan analisis secara global dari keseluruhan hasil pengujian.

6.1 Pengujian dan Analisis Pengaruh Generasi

Pada pengujian pertama ini bertujuan untuk mengetahui pengaruh perubahan dari jumlah generasi terhadap nilai evaluasi sistem yang dihasilkan. Jumlah generasi yang digunakan dalam pengujian ini adalah 30, 60, 90, 120, 150, 180, 210, 240, 270, dan 300. Setiap jumlah generasi akan dilakukan pengujian terhadap data uji sebanyak sepuluh kali dengan jumlah populasi sebanyak 10 dan nilai parameter *cr* (0,4) dan *mr* (0,6). Dari sepuluh kali proses pengujian akan dilakukan rata-rata untuk tiap jumlah generasi untuk menemukan jumlah generasi yang terbaik. Dalam pengujian ini didapatkan nilai rata-rata yang berbeda-beda. Hasil dari pengujian pengaruh jumlah generasi dapat dilihat pada Tabel 6.1.

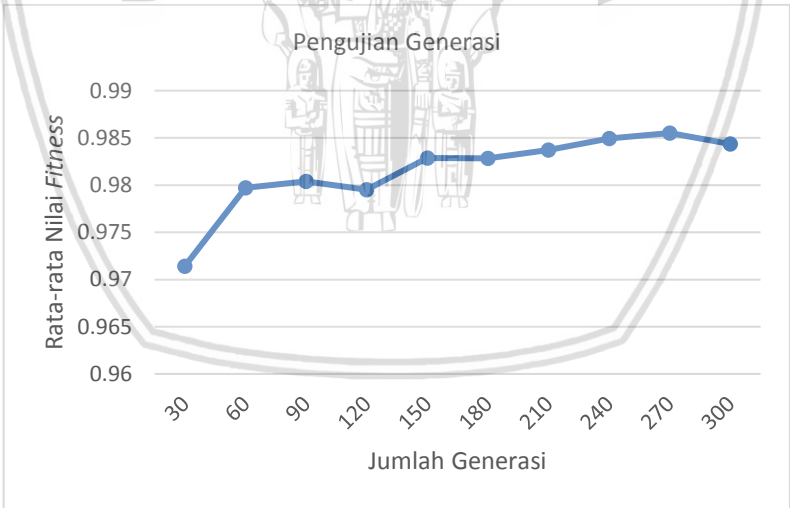
Tabel 6.1 Hasil Pengujian Pengaruh Generasi

Uji Coba Ke-	Generasi									
	30	60	90	120	150	180	210	240	270	300
1	0.980392	0.980392	0.981354	0.980392	0.979432	0.977517	0.981354	0.974659	0.98912	0.978474
2	0.971503	0.979432	0.981354	0.980392	0.987167	0.982318	0.977517	0.985222	0.988142	0.980392
3	0.971817	0.979432	0.979432	0.978474	0.980392	0.986193	0.982318	0.980392	0.982318	0.983284
4	0.970874	0.982318	0.975134	0.97371	0.982318	0.986193	0.986193	0.986193	0.984091	0.988142
5	0.971817	0.979432	0.981354	0.978474	0.988142	0.976563	0.984252	0.988142	0.98912	0.984252

6	0.97 0874	0.97 4659	0.98 7947	0.97 9432	0.98 3043	0.97 2763	0.98 8142	0.99 0099	0.98 3284	0.98 912
7	0.96 8992	0.98 3284	0.97 8234	0.98 2318	0.97 9432	0.98 7167	0.97 9432	0.98 3284	0.97 8474	0.98 0392
8	0.97 1817	0.97 4659	0.98 0152	0.98 3091	0.98 4252	0.98 4252	0.98 6193	0.98 5222	0.98 8142	0.99 0099
9	0.96 8992	0.98 2157	0.97 7517	0.98 0392	0.98 2125	0.98 7167	0.98 8631	0.98 912	0.98 8142	0.98 8142
10	0.96 7118	0.98 1354	0.98 1354	0.97 8474	0.98 2318	0.98 8142	0.98 3284	0.98 7167	0.98 4252	0.98 1354
Rata-rata	0.97 142	0.97 9712	0.98 0383	0.97 9515	0.98 2862	0.98 2828	0.98 3732	0.98 495	0.98 5508	0.98 4365

Berdasarkan Tabel 6.1 nilai *fitness* tertinggi didapat dari jumlah generasi sebanyak 270 dengan rata-rata nilai *fitness* sebesar 0.985508. Sedangkan rata-rata nilai *fitness* terendah didapat dari jumlah generasi sebanyak 30 dengan rata-rata nilai *fitness* sebesar 0.97142. Peningkatan rata-rata nilai *fitness* berbanding lurus dengan jumlah generasi.

Berikut ini merupakan grafik hubungan antara pengaruh jumlah generasi dengan hasil evaluasi sistem telah disajikan pada Gambar 6.1.



Gambar 6.1 Grafik Hasil Pengujian Pengaruh Generasi

Berdasarkan Gambar 6.1 menunjukkan bahwa jumlah generasi terbaik sebesar 270 karena memiliki rata-rata nilai *fitness* tertinggi. Peningkatan rata-rata nilai *fitness* dari generasi 10 sampai dengan 270. Akan tetapi terjadi penurunan pada generasi 120 dan 300. Hal ini dapat terjadi karena kromosom dibangkitkan secara *random*. Jika dilihat secara keseluruhan jumlah generasi berbanding lurus dengan niali *fitness*, semakin banyak jumlah generasi semakin tinggi pula nilai *fitness* yang dihasilkan. Akan tetapi apabila telah menemukan nilai *fitness* optimal pada jumlah generasi tertentu, maka nilai *fitness* untuk

generasi selanjutnya cenderung tetap atau stabil karena area pencarian sudah mencapai area optimal (Megantara, et al., 2017).

6.2 Pengujian dan Analisis Pengaruh *PopSize*

Pada pengujian kedua ini bertujuan untuk mengetahui pengaruh perubahan dari jumlah *popSize* terhadap nilai evaluasi sistem yang dihasilkan. Nilai *popSize* yang digunakan dalam pengujian ini adalah 10, 20, 30, 40, 50, 60, 70, 80, 90, dan 100. Setiap nilai *popSize* akan dilakukan pengujian terhadap data uji sebanyak sepuluh kali dengan jumlah generasi terbaik dari pengujian sebelumnya dan nilai parameter *cr* (0,4) dan *mr* (0,6). Dari sepuluh kali proses pengujian akan dilakukan rata-rata untuk tiap jumlah *popSize* yang diganti untuk menemukan jumlah *popSize* yang terbaik. Dalam pengujian ini didapatkan nilai rata-rata yang berbeda-beda. Hasil dari pengujian pengaruh jumlah *popSize* dapat dilihat pada Tabel 6.2.

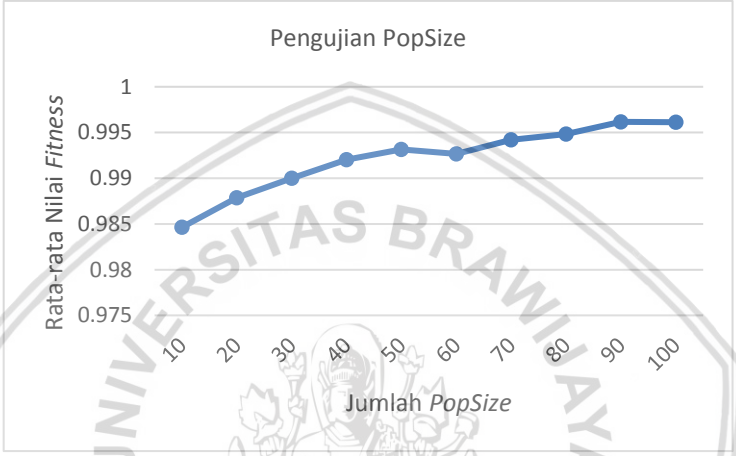
Berdasarkan Tabel 6.2 nilai *fitness* tertinggi didapat dari jumlah populasi sebanyak 90 dengan rata-rata nilai *fitness* sebesar 0.996168. Sedangkan rata-rata nilai *fitness* terendah didapat dari jumlah populasi sebanyak 10 dengan rata-rata nilai *fitness* sebesar 0.984635. Nilai *fitness* yang dihasilkan berbeda-beda karena nilai gen pada kromosom, penentuan *parent*, serta proses reproduksi yang didapat secara *random* sehingga tidak akan sama pada setiap perulangannya.

Tabel 6.2 Hasil Pengujian Pengaruh *PopSize*

Uji Coba Ke-	*PopSize*									
	10	20	30	40	50	60	70	80	90	100
1	0.986193	0.984252	0.98912	0.99108	0.996016	0.995025	0.994036	0.999001	0.993049	1
2	0.984252	0.99108	0.987167	0.994036	0.98912	0.995025	0.996016	0.996016	0.998004	0.994036
3	0.988142	0.99108	0.98912	0.98912	0.995025	0.99108	0.996678	0.994036	0.99552	0.998004
4	0.987947	0.992063	0.992063	0.994036	0.996016	0.993049	0.995025	0.997009	0.994036	0.995025
5	0.982318	0.987167	0.99108	0.993049	0.993049	0.990099	0.994036	0.994036	0.998004	0.996016
6	0.98912	0.985222	0.993049	0.988142	0.993049	0.998004	0.997009	0.993049	0.998004	0.997009
7	0.97561	0.987167	0.99108	0.995025	0.995025	0.990099	0.99108	0.995025	0.998004	0.994036
8	0.985222	0.987167	0.992063	0.991572	0.995025	0.99108	0.992063	0.990099	0.997009	0.995025

9	0.98 6193	0.98 6193	0.98 7167	0.99 108	0.99 4036	0.99 108	0.99 6016	0.99 5025	0.99 6016	0.99 6016
10	0.98 1354	0.98 7167	0.98 8142	0.99 3049	0.98 5222	0.99 2063	0.99 0099	0.99 5025	0.99 4036	0.99 6016
Rata-rata	0.98 4635	0.98 7856	0.99 0005	0.99 2019	0.99 3158	0.99 266	0.99 4206	0.99 4832	0.99 6168	0.99 6118

Berikut ini merupakan grafik hubungan antara pengaruh jumlah *popSize* dengan hasil evaluasi sistem telah disajikan pada Gambar 6.2.



Gambar 6.2 Grafik Hasil Pengujian Pengaruh *PopSize*

Berdasarkan Gambar 6.2 menunjukkan bahwa jumlah populasi terbaik sebesar 90 karena memiliki rata-rata nilai *fitness* tertinggi. Peningkatan rata-rata nilai *fitness* dari populasi 10 sampai dengan 90. Akan tetapi pada percobaan jumlah populasi 50 dan 60 mengalami penurunan rata-rata nilai *fitness* dari 0.993158 menjadi 0.99266. Dan pada percobaan jumlah populasi 90 dan 100 mengalami penurunan rata-rata nilai *fitness* dari 0.996168 menjadi 0.996118. Jumlah populasi yang besar tidak menjamin akan menghasilkan nilai *fitness* yang lebih tinggi, hal tersebut dapat terjadi karena pembangkitan populasi awal yang didapat secara *random* (Kusumaningsih, 2016). Besar jumlah populasi berdampak pada keberagaman nilai yang dihasilkan, karena semakin banyak ragam individu yang dihasilkan dari proses reproduksi (Kusumaningsih, 2016).

6.3 Pengujian dan Analisis Pengaruh Parameter Reproduksi

Pada proses pengujian ketiga dilakukan untuk mengetahui pengaruh parameter reproduksi terhadap hasil evaluasi sistem yang telah diimplementasikan. Dalam pengujian ini dilakukan dengan nilai *cr* dan *mr* yang berbeda-beda yakni, nilai *cr* 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, dan 0.9. Sedangkan untuk nilai *mr* 0,9. 0.8, 0.7, 0.6, 0.5, 04, 0.3, 02, 0.1. Pengujian ini bertujuan untuk mendapatkan nilai *cr* dan *mr* terbaik, sehingga akan memudahkan pada proses komputasi. Setiap nilai *cr* dan *mr* akan dilakukan pengujian terhadap data uji sebanyak sepuluh kali dengan jumlah generasi dan

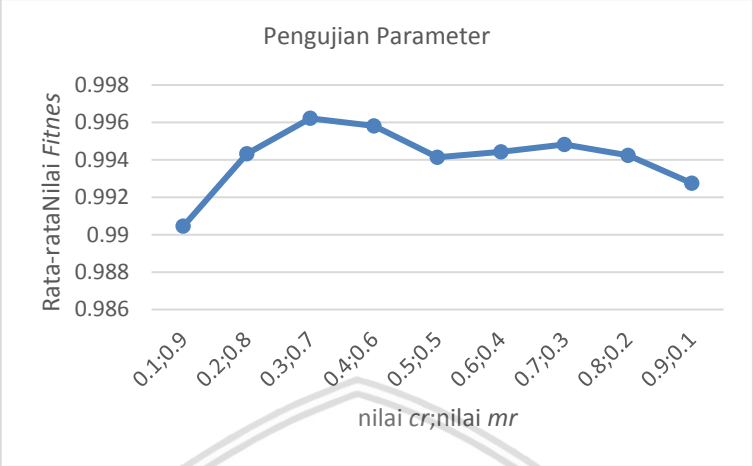
popSize terbaik dari pengujian sebelumnya. Dari sepuluh kali proses pengujian akan dilakukan rata-rata untuk tiap nilai *cr* dan *mr* yang diganti untuk menemukan nilai *cr* dan *mr* yang terbaik. Dalam pengujian ini didapatkan nilai rata-rata yang berbeda-beda. Untuk nilai hasil evaluasi pengujian pengaruh *cr* dan *mr* dapat dilihat pada Tabel 6.3.

Tabel 6.3 Hasil Pengujian Pengaruh Parameter Reproduksi

Uji Coba Ke-	Parameter (*cr*, *mr*)								
	0.1; 0.9	0.2; 0.8	0.3; 0.7	0.4; 0.6	0.5; 0.5	0.6; 0.4	0.7; 0.3	0.8; 0.2	0.9; 0.1
1	0.987167	0.994036	0.997009	0.995025	0.995025	0.993049	0.996016	0.994036	0.992063
2	0.997506	0.994036	0.997009	0.998004	0.996016	0.993049	0.995025	0.993049	0.996016
3	0.98912	0.995025	0.995025	0.996016	0.995025	0.995025	0.995025	0.994036	0.993049
4	0.992063	0.996016	0.995025	0.995025	0.995025	0.994036	0.993049	0.995025	0.990099
5	0.988142	0.999001	0.99108	0.993049	0.993049	0.994036	0.992063	0.992063	0.992063
6	0.992063	0.98912	0.997009	0.997009	0.993049	0.994036	0.995025	0.996016	0.993049
7	0.990099	0.99108	0.996016	0.996016	0.995025	0.994036	0.995025	0.998004	0.990099
8	0.988142	0.990884	0.998004	0.997009	0.993049	0.996016	0.996016	0.990099	0.993049
9	0.992063	0.996016	0.996016	0.995025	0.99108	0.998004	0.998004	0.994036	0.995025
10	0.988142	0.998004	1	0.996016	0.995025	0.993049	0.993049	0.996016	0.993049
Rata-rata	0.990451	0.994322	0.996219	0.995819	0.994137	0.994433	0.99483	0.994238	0.992756

Berdasarkan Tabel 6.3 nilai *fitness* tertinggi didapat dari nilai *cr* dan *mr* yang bernilai 0.3 dan 0.7 dengan rata-rata nilai *fitness* sebesar 0.996219. Sedangkan rata-rata nilai *fitness* terendah didapat dari nilai *cr* dan *mr* yang bernilai 0.1 dan 0.9 dengan rata-rata nilai *fitness* sebesar 0.990451.

Berikut ini adalah grafik hubungan antara pengaruh parameter reproduksi dengan hasil evaluasi sistem telah disajikan pada grafik pengaruh parameter reproduksi pada Gambar 6.3 dibawah ini:



Gambar 6.3 Grafik Hasil Pengujian Pengaruh Parameter Reproduksi

Berdasarkan Gambar 6.3 menunjukkan rata-rata nilai *fitness* tertinggi sebesar 0.996219 pada nilai *cr* 0.3 dan nilai *mr* 0.7. Hasil pengujian pengaruh parameter reproduksi tidak menunjukkan keseluruhan nilai yang stabil. Nilai *crossover rate* yang terlalu rendah dan nilai *mutation rate* yang terlalu tinggi akan menurunkan kemampuan eksplorasi algoritme genetika dalam pencarian. Nilai *crossover rate* yang terlalu tinggi dan nilai *mutation rate* yang terlalu rendah akan menurunkan tingkat pembelajaran algoritme genetika dalam hal belajar dari generasi sebelumnya (Sari, et al., 2014). Oleh karena itu dalam menentukan kombinasi nilai *cr* dan *mr* yang tepat tidak mudah, bergantung pada permasalahan yang diangkat (Mahmudy, 2015).

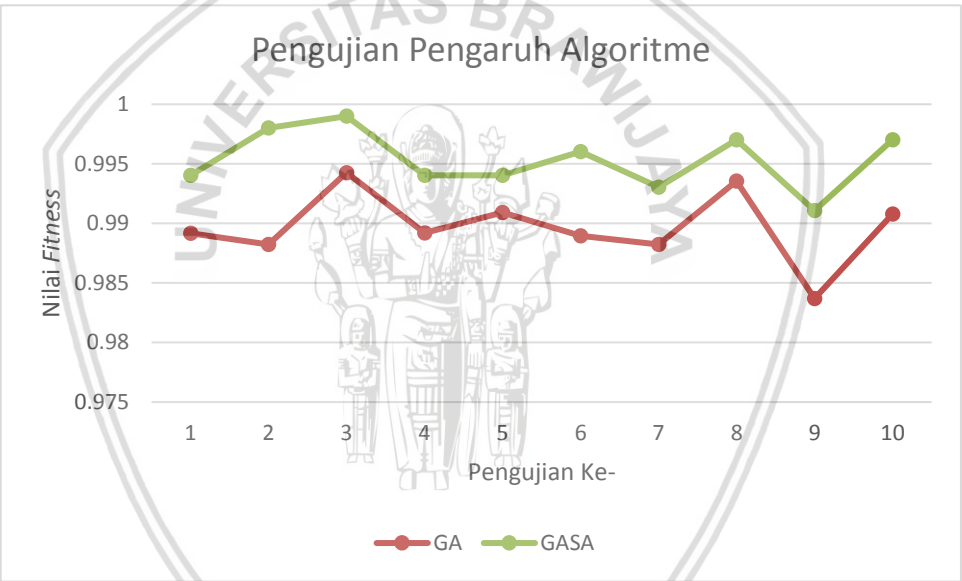
6.4 Pengujian dan Analisis Pengaruh Algoritme

Pada pengujian keempat ini bertujuan untuk mengetahui apakah hibridisasi algoritme GA-SA lebih baik dibanding dengan algoritme genetika murni terhadap nilai evaluasi sistem yang dihasilkan. Setiap perbandingan algoritme akan dilakukan pengujian terhadap data uji sebanyak sepuluh kali dengan parameter jumlah generasi 270, jumlah populasi 90, nilai *cr* 0.3, dan nilai *mr* 0.7. Dari sepuluh kali proses pengujian akan dilakukan rata-rata nilai perbandingan untuk mengetahui apakah hibridisasi algoritme GA-SA lebih baik dibanding dengan algoritme genetika murni. Dalam pengujian ini didapatkan nilai rata-rata yang berbeda. Hasil dari pengujian pengaruh jumlah generasi dapat dilihat pada Tabel 6.4.

Berdasarkan Tabel 6.4 nilai rata-rata *fitness* algoritme GA-SA sebesar 0.995328 lebih tinggi dibanding nilai rata-rata *fitness* algoritme GA saja sebesar 0.989699. Selisih nilai keduanya sebesar 0,00563. Gambar 6.4 merupakan grafik hubungan antara pengaruh hibridisasi algoritme GA-SA dengan algoritme genetika murni.

Tabel 6.4 Hasil Pengujian Pengaruh Algoritme

Percobaan Ke-	GA	GA-SA
1	0.989172	0.994036
2	0.988222	0.998004
3	0.994263	0.999001
4	0.989192	0.994036
5	0.990927	0.994036
6	0.988947	0.996016
7	0.988221	0.993049
8	0.993554	0.997009
9	0.983694	0.99108
10	0.990796	0.997009
Rata-Rata	0.989699	0.995328



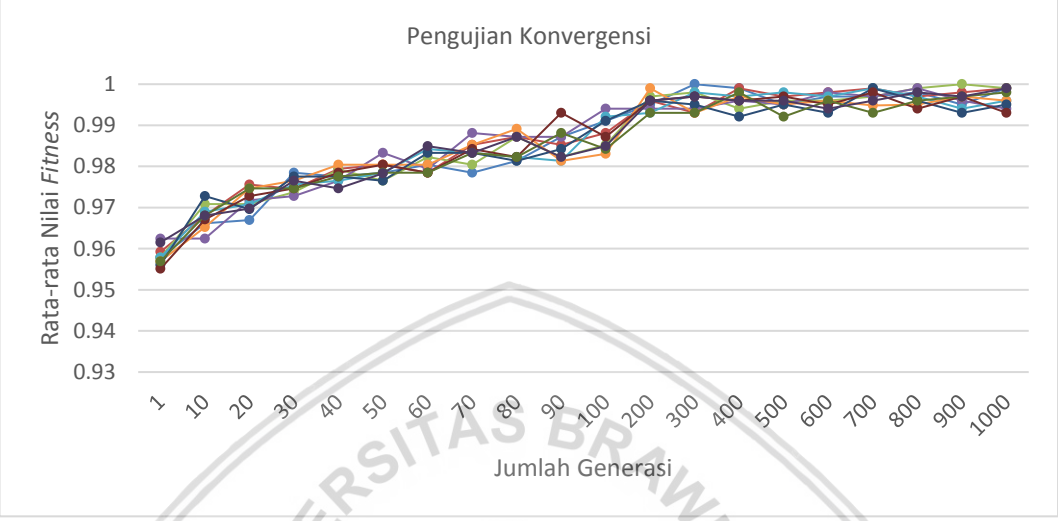
Gambar 6.4 Grafik Hasil Pengujian Pengaruh Algoritme

Berdasarkan Gambar 6.4 nilai *fitness* yang dihasilkan dari hibridisasi algoritme GA-SA lebih besar dibandingkan dengan nilai *fitness* algortime genetika murni. Kondisi tersebut dipengaruhi oleh adanya modifikasi individu terbaik dari hasil algoritme genetika murni dengan menggunakan metode *neighborhood move* pada iterasi SA yang berfungsi untuk meningkatkan eksplorasi supaya tidak terjebak pada *local* optimum.

6.5 Pengujian Konvergensi

Pengujian konvergensi dilakukan untuk mengetahui pada generasi keberapa nilai *fitness* yang dihasilkan sudah tidak mengalami perbedaan yang signifikan. Dapat dikatakan konvergen apabila nilai *fitness* yang dihasilkan stabil, artinya rata-rata selisih nilai *fitness* tidak lebih dari 0.01.

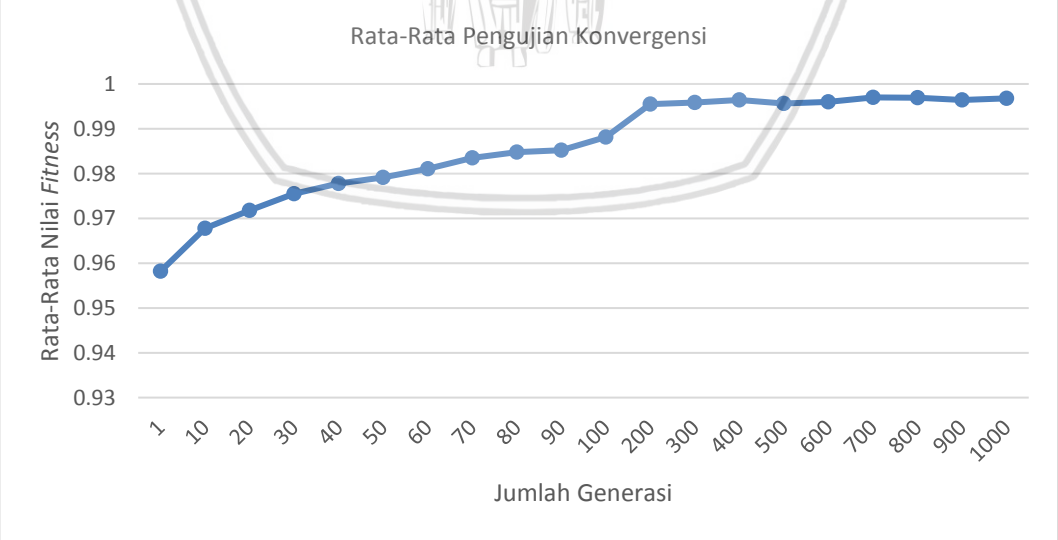
Parameter yang digunakan merupakan parameter terbaik dari hasil pengujian sebelumnya, yaitu jumlah populasi sebanyak 90, nilai *cr* sebesar 0.3 dan nilai *mr* sebesar 0.7. Berikut ini merupakan grafik hasil pengujian konvergensi dengan jumlah populasi sebanyak 90, nilai *cr* sebesar 0.3 dan nilai *mr* sebesar 0.7 telah disajikan pada Gambar 6.5.



Gambar 6.5 Grafik Hasil Pengujian Konvergensi

Berdasarkan Gambar 6.6 dapat dilihat bahwa nilai *fitness* yang dihasilkan kovergen atau sudah tidak mengalami perbedaan yang signifikan pada generasi ke-200. Warna-warna di atas menunjukkan banyaknya pengujian yang dilakukan sebanyak sepuluh kali pada setiap generasinya. Dan rata-rata nilai *fitness* pada setiap generasi dapat dilihat pada Gambar 6.7.

Gambar 6.6 Grafik Hasil Rata-Rata Pengujian Konvergensi



6.6 Analisis Global dari Keseluruhan Hasil Pengujian

Berdasarkan perancangan, implementasi, pengujian, serta analisis sebelumnya dapat dikatakan bahwa hibridisasi algoritme genetika dan *simulated*

annealing dapat digunakan untuk mengoptimasi penjadwalan mata pelajaran yang ada di SMA Negeri 6 Surabaya. Berikut adalah beberapa tahapan yang dilakukan dalam mengimplementasikan sistem, antara lain:

1. Inisialisasi kromosom, yaitu membentuk individu awal dengan membangkitkan nilai dari data kode tugas mengajar. Panjang kromosom didapat dari jumlah keseluruhan hasil pengurangan antara jumlah jam pelajaran dari hari senin-jumat dengan total jam pada masing-masing tingkatan kelas dan ditambah dengan slot kosong masing-masing tingkatan kelas. Dalam penelitian ini panjang kromosom sepanjang 378. Setiap kromosom yang diinisialisasi merupakan representasi dari solusi permasalahan.
2. Reproduksi, proses ini terbagi menjadi dua yaitu *crossover* dan mutasi. Teknik *crossover* yang digunakan adalah one cut point *crossover* dan mutasi yang digunakan adalah *reciprocal* exchange *mutation*. Pada proses reproduksi ini akan menghasilkan keturunan (*child*).
3. Evaluasi, yaitu proses menghitung nilai *fitness* pada setiap individu baik sebagai *parent* atau *child*. Nilai *fitness* didapatkan dari berapa banyak pelanggaran yang terjadi di setiap individu.
4. Seleksi merupakan proses untuk memilih individu dengan nilai *fitness* terbaik dari keseluruhan individu sejumlah populasi awal untuk diproses pada generasi selanjutnya.
5. *Simulated annealing*, pada proses ini akan dipilih satu individu terbaik hasil seleksi pada algoritme genetika untuk dilakukan penukaran dua nilai gen yang berbeda dan individu baru (*neighborhood move*).

Sistem penjadwalan mata pelajaran menggunakan hibridisasi algoritme genetika dan *simulated annealing* yang dijalankan dengan menggunakan beberapa parameter terbaik hasil dari pengujian sebelumnya, yaitu generasi terbaik berjumlah 270 generasi, populasi terbaik sebesar 90 individu, nilai *cr*, dan *mr* sebagai parameter reproduksi menghasilkan nilai 0,3 dan 0,7. Dengan nilai-nilai tersebut diperoleh nilai *fitness* sebesar 0.999000 dengan jumlah pelanggaran sebanyak 1.

Jadwal yang dihasilkan oleh sistem telah mendekati optimal, akan tetapi masih ada pelanggaran yang terjadi. Namun hasil penjadwalan ini dapat digunakan, karena tidak adanya pelanggaran yang fatal yaitu pelanggaran ke-1 mengenai jadwal yang bentrok. Tabel 6.5 menunjukkan detail hasil pelanggaran *constraint* dari jadwal yang dihasilkan oleh sistem.

Berdasarkan Tabel 6.5 pelanggaran terjadi pada mata pelajaran Penjaskes OR berada di jam ke-5 (dapat dilihat pada Lampiran), hal ini masih dapat di toleransi. Karena dari hasil wawancara pada lampiran menyatakan bahwa, mata pelajaran Penjaskes OR berada pada jam pelajaran ke 1-4 akan tetapi jika tidak memungkinkan dapat diletakkan pada jam pelajaran ke 5-6.

Penulis melakukan perbandingan terhadap jadwal mata pelajaran yang telah dibuat secara manual oleh pihak sekolah. Jadwal yang dibuat secara manual tidak terdapat pelanggaran, baik pelanggaran terhadap *hard constraint* maupun *soft constraint.* Detail pelanggaran dapat dilihat pada Tabel 6.6.

Tabel 6.5 Detail Pelanggaran *Constraint* Hasil Sistem

NO.	*CONSTRAINT*	Jumlah Pelanggaran
1	Guru yang mengajar tidak boleh mengajar lebih dari satu kelas atau mengajar di lain kelas pada waktu yang sama (bentrok)	0
2	Mata pelajaran Penjaskes OR hanya boleh berada pada jam pelajaran ke 1-4	1
3	Guru yang mengajar tidak boleh mengajar lebih dari empat jam pelajaran dalam mata pelajaran yang sama dan di hari yang sama	0
4	Dalam satu mata pelajaran yang sama pada jam kedua dan selanjutnya (sejumlah ketentuan) tidak boleh beda hari	0
Total Pelangaran		1
Nilai *Fitness*		0.999000

Tabel 6.6 Detail Pelanggaran *Constraint* Jadwal Manual

NO.	*CONSTRAINT*	Jumlah Pelanggaran
1	Guru yang mengajar tidak boleh mengajar lebih dari satu kelas atau mengajar di lain kelas pada waktu yang sama (bentrok)	0
2	Mata pelajaran Penjaskes OR hanya boleh berada pada jam pelajaran ke 1-4	0
3	Guru yang mengajar tidak boleh mengajar lebih dari empat jam pelajaran dalam mata pelajaran yang sama dan di hari yang sama	0
4	Dalam satu mata pelajaran yang sama pada jam kedua dan selanjutnya (sejumlah ketentuan) tidak boleh beda hari	0
Total Pelangaran		0
Nilai *Fitness*		1

Berdasarkan Tabel 6.5 nilai *fitness* hasil penjadwalan yang dihasilkan oleh sistem hampir mendekati optimal dan belum optimal seperti penjadwalan yang dilakukan secara manual, hal ini dipengaruhi oleh beberapa faktor. Pertama, data yang dijadwalkan tergolong besar dengan 1080 slot jadwal, 41 guru, dan 19 mata pelajaran. Kedua, inisialisasi kromosom awal yang dibangkitkan secara *random* tidak dapat menjamin bahwa individu yang yang dihasilkan memiliki susunan gen yang baik. Ketiga, pemilihan *parent* pada proses reproduksi juga dilakukan secara *random* dan juga tidak menjamin bahwa *parent* tersebut akan menghasilkan individu yang lebih baik.

Ditinjau dari nilai *fitness* yang dihasilkan, nilai *fitness* hasil penjadwalan manual lebih baik daripada nilai *fitness* hasil penjadwalan dengan sistem. Akan tetapi jika ditinjau dari waktu yang dibutuhkan dalam menyusun jadwal mata pelajaran, penjadwalan dengan sistem membutuhkan waktu lebih singkat ± 20 detik. Sedangkan waktu yang dibutuhkan dalam menyusun jadwal secara manual membutuhkan ± 1 minggu.

BAB 7 PENUTUP

Pada bab ini berisi kesimpulan yang didapatkan setelah melakukan proses penelitian dan saran sebagai masukan untuk penelitian selanjutnya.

7.1 Kesimpulan

Berdasarkan penelitian yang telah dilakukan mengenai penjadwalan mata pelajaran, dapat diambil beberapa kesimpulan sebagai berikut:

1. Hibridisasi algortime genetika dan *simulated annealing* dapat diimplementasikan untuk mengoptimasi penjadwalan mata pelajaran di SMA Negeri 6 Surabaya. Implementasi tersebut melalui beberapa proses antara lain, inisialisasi kromosom, reproduksi, evaluasi, seleksi, serta *neighborhood move* pada *simulated annealing*. Kromosom direpresentasikan dengan bilangan integer yang berisi kode tugas mengajar dan kode mata pelajaran. Panjang kromosom sejumlah 378 berasal dari jumlah keseluruhan hasil pengurangan antara jumlah jam pelajaran dengan total jam pada masing-masing tingkatan kelas dan ditambah dengan slot masing-masing tingkatan kelas. Setiap kromosom yang diinisialisasi merupakan representasi dari solusi permasalahan. Solusi permasalahan tersebut memiliki nilai *fitness*, nilai *fitness* tertinggi merupakan solusi yang optimal karena memiliki jumlah pelanggaran *constraint* terkecil. Metode yang digunakan dalam proses reproduksi adalah one cut point *crossover* dan *reciprocal exchange mutation*. Serta metode seleksi yang digunakan adalah *elitism* dan modifikasi solusi pada SA menggunakan *neighborhood move*.
2. Berdasarkan pengujian sebelumnya menghasilkan generasi terbaik sejumlah 270, populasi terbaik sebanyak 90, nilai *cr* sebesar 0.3, dan nilai *mr* sebesar 0.7 dengan nilai *fitness* sebesar 0.999000 dan variabel konstan yaitu koefisien penurunan suhu (*β*) bernilai 0.5, suhu awal (*Tc*) bernilai 15, dan suhu akhir (*Ta*) bernilai 10. Solusi yang dihasilkan oleh sistem cukup baik, karena nilai *fitness* yang diperoleh mendekati optimal. Namun tidak lebih baik dibanding penjadwalan manual yang optimal tanpa ada pelangaran. Akan tetapi jika ditinjau dari segi waktu, penjadwalan dengan sistem lebih baik, karena hanya membutuhkan waktu ± 20 detik dibandingkan dengan penjadwalan manual yang membutuhkan waktu ± 1 minggu.

7.2 Saran

Dari hasil pengujian dan implementasi yang dilakukan dalam penelitian ini, penulis memberi beberapa saran yang dapat diterapkan pada penelitian selanjutnya. Beberapa saran akan dijelaskan sebagai berikut:

1. Sebelum memasuki proses reproduksi, sebaiknya dilakukan teknik pemilihan *parent* agar individu yang dihasilkan merupakan individu dari gen yang terbaik.

2. Penentuan titik potong *crossover* juga mempengaruhi, titik potong yang dilakukan pada gen yang bermasalah akan mempermudah proses mendpatkan individu dengan pelanggaran yang lebih sedikit. Seperti halnya *crossover*, titik potong mutasi juga mempengaruhi. Apabila dilakukan penukaran gen pada gen yang bermasalah, maka akan mempercepat waktu komputasi dan menghasilkan nilai *fitness* yang lebih baik.
3. Menggunakan nilai *cr* dan *mr* yang adaptif, artinya pada iterasi awal menggunakan nilai *cr* tinggi dan *mr* rendah untuk pencarian solusi seluas-luasnya. Sedangkan pada iterasi mendekati akhir menggunakan nilai *cr* rendah dan *mr* tinggi untuk pencarian solusi sedalam-dalamnya.
4. Menggunakan operator modifikasi solusi yang tidak terlalu mengandalkan nilai *random*, seperti *weighted neighborhood move* yaitu penukaran dua buah gen yang memiliki pelanggaran yang terbanyak.
5. Menambahkan nilai untuk variable konstan seperti koefisien penurunan suhu (β), suhu awal (*Tc*), dan suhu akhir (*Ta*) pada proses GA-SA untuk menambah pencarian solusi yang lebih beragam dan menambahkan pengujian untuk parameter *simulated annealing* yaitu koefisien penurunan suhu (β), suhu awal (*Tc*), dan suhu akhir (*Ta*).

DAFTAR PUSTAKA

Adamanti, J., 2002. *Penyelesaian Masalah Penjadwalan Mata Kuliah di Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Gadjah Mada dengan Menggunakan Algoritma Genetika,* Jogjakarta: Universitas Gadjah Mada.

Al-Milli, N. R., 2011. Hybrid Genetic Algorithm with Simulated Annealing For University Course Timetabling Problems. *Journal of Theoretical and Applied Information Technology,* 29(2).

Avicena, A., 2016. *Optimasi Penjadwalan Pengawas Ujian Semester Menggunakan Hibridasi Algoritma Genetika Dan Simulated Annealing (Studi Kasus : Fakultas Ilmu Komputer Universitas Brawijaya),* Malang: Fakultas Ilmu Komputer Universitas Brawijaya.

Benli, O. S. & Botsali, A. R., 2004. *Timetabling problem.* s.l.:s.n.

Bertsimas, D. & Tsitsiklis, J., 1998. Simulated Annealing. Dalam: *Statistical Science, 8(1).* s.l.:s.n., pp. 10-15.

Burke, E. K., Newall, J. P. & Weare, R. F., 1996. *A Memetic Algorithm for University Exam Timetabling,* s.l.: University of Nottingham.

Darmawan, A. & Hasibuan, R. M., 2014. *Penjadwalan Mata Kuliah Menggunakan Algoritma Genetika dengan Mempertimbangkan Team-teaching,* Yogyakarta: Universitas Gadjah Mada.

Efendi, M. F., Cholissodin, I. & Santoso, E., 2017. Optimasi Penjadwalan Mata Pelajaran Menggunakan Algoritma Genetika (Studi Kasus: SMK Negeri 2 Kediri). *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer ,* 1(10), pp. 1066-1072.

Elhaddad, Y. R., 2012. *Combined Simulated Annealing and Genetic Algorithm to Solve Optimization Problems,* Libya: World Academy of Science.

Elhaddad, Y. & Sallabi, O., 2010. *A New Hybrid Genetic and Simulated Annealing Algorithm to Solve the Traveling Salesman Problem,* London: WCE.

Gendreau, M. & Potvin, J. -Y., 2010. *Handbook of Metaheuristics.* 2nd penyunt. New York: Springer.

Gen, M. & Cheng, R., 1997. *Genetic Algorithms and Engineering Design,* New York: John Wiley & Sons, Inc.

Glover, F. W. & Kochenberger, G. A., 2003. *Handbook of Metaheuristics,* Dordrecht: Kluwer Academic Publishers.

Gultom, S., 2013. *Pedoman Pelatihan Implementasi Kurikulum 2013.* Jakarta: Badan Pengembangan Sumber Daya Manusia Pendidikan dan Kebudayaan dan Penjaminan Mutu Pendidikan.

Ilmi, R. R., 2015. *Optimasi Penjadwalan Perawat Menggunakan Algoritma Genetika,* Malang: Fakultas Ilmu Komputer Universitas Brawijaya.

Iman, F., Ratnawati, D. E. & Kusuma, T. S., 2018. Optimasi Komposisi Makanan Untuk Ibu Hamil Menggunakan Hybrid Algoritme Genetika dan Simulated Annealing. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer ,* 2(11), pp. 4325-4332.

Kirkpatrick, S., Gelatt, C. D. & Vecchi, M. P., 1983. *Optimization by Simulated Annealing,* s.l.: Science.

Kusumaningsih, F. D., 2016. *Penerapan Algoritma Genetika Pada Optimasi Susunan Bahan Makanan Untuk Pemenuhan Kebutuhan Gizi Keluarga,* Malang.: Fakultas Ilmu Komputer Universitas Brawijaya.

Lin, S.-W., Yu, V. F. & Chou, S.-Y., 2011. A Simulated Annealing Heuristic for the Truck and Trailer Routing Problem with Time Windows. *Expert Systems with Applications,* Volume 38, pp. 15244-15252.

Mahmudy, W. F., 2013. *Algoritma Evolusi,* Malang: Universitas Brawijaya.

Mahmudy, W. F., 2015. *Dasar-Dasar Algoritma Evolusi,* Malang, Indonesia: Program Teknik Informasi dan Ilmu Komputer, Universitas Brawijaya.

Mahmudy, W. F., Marian, R. M. & Luong, L. H. S., 2013. Hybrid Genetic Algorithms for Part Type Selection and Machine Loading Problems with Alternative Production Plans in Flexible Manufacturing System. *IEEE International Conferenceon Computational Innteligence and Cybernetics,* pp. 126-130.

Mawaddah, N. K. & Mahmudy, W. F., 2006. *Optimasi Penjadwalan Ujian Menggunakan Algoritma Genetika,* Malang: Jurusan Matematika FMIPA Universitas Brawijaya.

Megantara, A. N., Setiawan, B. D. & Wihandika, R. C., 2017. Optimasi Fuzzy Inference System Mamdani Menggunakan Algoritme Genetika untuk Menentukan Lama Waktu Siram pada Tanaman Strawberry. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer,* Volume 1, pp. 1433-1442.

Norozi, A. A., M., K. A., Ismail, N. & Mustapha, F., 2011. An Optimization Technique Using Hyrid GA-SA Algorithm for Multi-objective Scheduling Problem. Dalam: *Scientific Research and Essays, Volume 6(8).* s.l.:s.n., pp. 1720-1731.

Norozi, A., Ariffin, M., Ismail, N. & Musthapa, F., 2012. A hybrid GA-SA Algorithm for Multi-objective Sequencing Problem in High-product Mix Shop-floor. *Applied Mechanics and Materials,* Volume 110-116, pp. 3964-3971.

Poerwati, L. E. & Amri, S., 2013. *Panduan Memahami Kurikulum 2013.* Jakarta: PT. Prestasi Pustakarya.

Pratiwi, R. N., 2017. *Optimasi Penjadwalan Mata Pelajaran Pada Kurikulum 2013 Dengan Algoritma Genetika (Studi Kasus : SMA Negeri 3 Surakarta),* Malang: Fakultas Ilmu Komputer Universitas Brawijaya.

Sari, A. P., Mahmudy, W. F. & Dewi, C., 2014. Optimasi Asupan Gizi Pada Ibu Hamil Dengan Menggunakan Algoritma Genetika. *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya,* Volume 4, p. 5.

Sari, D. D. P., Mahmudy, W. F. & Ratnawati, D. E., 2015. Optimasi Penjadwalan Mata Pembelajaran Menggunakan Algoritma Menggunakan Algoritma Genetika (Studi Kasus : SMPN 1Gondang Mojokerto). *DORO : Reposiroty Jurnal Mahasiswa PTIIK Universitas Brawijaya, 5,* p. 13.

Seissarina, M. L., 2016. *Penjadwalan Pengawas Ujian Semester Menggunakan Algoritma Genetika,* Malang: Fakultas Ilmu Komputer Universitas Brawijaya.

Sivanandam, S. N. & Deepa, S. N., 2008. *Introduction to Genetic Algorithms,* New York: Springer Berlin Heidelberg.

Sofianti, T. D., 2004. *Penjadwalan Multipurpose batch Chemical Plant Dengan Metode Optimasi Gabungan : Algoritma Genetika - Simulated Annealing,* Jakarta: KOMMIT 2004 Auditorium Universitas Gunadarma.

Wen, S. & Wen-jun, S., 2006. Decision of Production Planning in Agricultural Management Using Genetic Algorithm. pp. 645-650.